Journal Pre-proof

Software Architectural Design in Global Software Development: An Empirical Study

Outi Sievi-Korte, Ita Richardson, Sarah Beecham

PII: S0164-1212(19)30174-8
DOI: https://doi.org/10.1016/j.jss.2019.110400
Reference: JSS 110400

To appear in: *The Journal of Systems & Software*

Received date: 12 December 2018
Revised date: 14 August 2019
Accepted date: 18 August 2019

Please cite this article as: Outi Sievi-Korte, Ita Richardson, Sarah Beecham, Software Architectural Design in Global Software Development: An Empirical Study, *The Journal of Systems & Software* (2019), doi: https://doi.org/10.1016/j.jss.2019.110400

Highlights

- A holistic view of architecting in GSD, combining recommendations from both literature and this empirical study.
- A set of 8 recommendations for how to conduct architectural practices in Global Software Development.
- A set of 8 challenges that act as warnings to those new to GSD.
- A visualisation of the relationships between the challenges and practices, and key themes (in UML).
- Quotes from a group of experts in the field, that highlight the problems other architects might relate to.

Software Architectural Design in Global Software Development: An Empirical Study

Outi Sievi-Korte[a], Ita Richardson[b], Sarah Beecham[b,*]

[a] *Tampere University, Finland*
[b] *Lero - The Irish Software Research Centre, University of Limerick, Ireland*

Abstract

In Global Software Development (GSD), the additional complexity caused by global distance requires processes to ease collaboration difficulties, reduce communication overhead, and improve control. How development tasks are broken down, shared and prioritized is key to project success.

While the related literature provides some support for architects involved in GSD, guidelines are far from complete. This paper presents a GSD Architectural Practice Framework reflecting the views of software architects, all of whom are working in a distributed setting. In-depth interviews with architects from seven different GSD organizations revealed a complex set of challenges and practices.

Designing software for distributed teams requires careful selection of practices that support understanding and adherence to defined architectural plans across sites. Teams used Scrum which aided communication, and Continuous Integration which helped solve synchronization issues. However, teams deviated from the design, causing conflicts. Furthermore, there needs to be a balance between the self-organizing Scrum team methodology and the need to impose architectural design decisions across distributed sites.

The research presented provides an enhanced understanding of architectural practices in companies using distributed development methods. Our GSD Architectural Practice Framework gives practitioners a cohesive set of warnings,

*Corresponding author
Email address: sarah.beecham@lero.ie (Sarah Beecham)

which for the most part, are matched by recommendations.

Keywords: software architecture, global software development, GSD, Scrum, GSE, empirical study

1. Introduction

Global software development (GSD) in its many forms has become a standard way of producing software for large companies [1] as well as small [2]. Tasks are outsourced and/or off-shored [3] for a variety of reasons, such as to reduce costs and gain access to local markets and resources [4]. No matter how tasks are distributed or what kind of processes are followed, there is one common denominator for all GSD projects that make them more challenging to handle than collocated projects, and that is 'global distance'.

Global distance [5] has three dimensions: socio-cultural, temporal and geographical. Geographical and temporal distance are a natural consequence of having development sites far away from each other. Socio-cultural distance can also cause problems with distributed development, due to issues of trust and misunderstandings [6].

Global distance thus calls for more effort in terms of inter and intra team communication, coordination and control [7]. Working communication methods need to be in place to overcome the challenges brought about by distance. Projects need to be especially well-coordinated [8], so that each site is at all times aware of their tasks and responsibilities and to ensure a common view of the status and requirements of the project [9, 10].

These GSD challenges can be alleviated by minimizing the need for communication between sites. This will ease task performance, lead to fewer meetings, fewer emails sent and fewer misunderstandings due to cultural differences. Herbsleb et al. [11] suggest that careful task allocation is key to achieving an optimal communication level, minimizing connections between sites. Tasks, and the connections between them, are derived directly from the dependencies within software, which are dictated by the software architecture.

Furthermore, Conway's law [12] states that the software architecture will end up mirroring the organization's communication structure, and this has been validated by many studies over the years [13, 14, 15]. Thus, it would seem that by creating a modular architecture that follows the organization's structure and available skills may solve a lot of issues with GSD, and the various barriers imposed by global distances [8].

Software architecture design, however, is a very complicated activity. In addition to reflecting on the modular structure of the software, architects need to consider required technologies and the dependencies between them, available resources, available budget and schedule, customer requirements and pressure from the marketing department, and such like. Particularly if there are concerns spanning multiple layers or various components, the modularity of a software itself is not straightforward either. For example, if tasks are divided by components, how can we handle features that require several components? And vice versa - if tasks are divided by features, how can we handle situations where several teams need the same component for their feature?

The overlapping nature of the two challenging aspects - GSD and software architecture design - is thus vital to investigate. What kind of practices exist to handle architecture design in a distributed environment? What are the recognized challenges and how are they handled? The importance of this intersection has already been noted by Babar and Lescher [16], who raise software architectural design as a key strategy for success in a GSD project.

A number of published studies highlight a range of architectural issues in a GSD context, e.g. [17, 18]. However, many of these studies present secondary results from synthesising or mapping architectural reviews and architectural knowledge management issues in GSD, without directly investigating how to perform software architecture design in a distributed setting. Further, while we found nine challenges and nine practices for architectural design in our SLR [19], the nine recommended practices only supported five of the challenges, leaving four without support. We found no solutions to challenges related to change management, quality control and development time task allocation. In this

empirical study we aim to resolve these gaps by interviewing practitioners in the field. Based on what we learned from our SLR [19], we are not expecting to discover practices that would be novel to the software architecture community as such, though, but to carve out a subset of practices shown to be important in a GSD context.

Taking a qualitative, inductive approach, we discovered yet more challenges to those observed in the literature, and were able to match known and new challenges with recommended practices which work in practice. These augmented sets of challenges and practices are captured in our GSD Architectural Practice Framework (Section 5).

This paper is organized as follows: Focusing on software architecting in GSD, section 2 presents the background. In Section 3 we outline our empirical research method and in Section 4 we summarize the results from the practitioner interviews. Section 5 presents unified practices and guidelines for software architecting in GSD - the GSD Architectural Practice Framework. In Section 6 we discuss our results and consider threats to validity. Finally, in Section 7, we summarize our contribution.

2. Background

2.1. Related Work

Software architecture related studies in a GSD context were reviewed by Mishra and Mishra [20] who viewed architecting in terms of either knowledge management (see, e.g., [21, 22, 23, 24]) or process and quality (see, e.g., [25, 26]). Additionally, there are several studies on performing software architecture reviews and evaluations in the context of GSD. Architecture reviews are an important part of quality and requirements management, as through them it can be verified that the architecture fulfills both functional and non-functional requirements. Such reviews are traditionally held in workshops and other face-to-face meetings, which are difficult to arrange in GSD projects. Ali Babar investigated the use and efficiency of tools to perform this task [27, 28]. Evaluation

of software architecture decisions, in turn, has been studied by Che and Perry [29].

Where architectural issues have been addressed in relation to task allocation and coordination of GSD projects, Conway's law features widely (see, e.g., [30, 31, 32, 33]). Herbsleb and Grinter [34], when discussing GSD, explicitly recommend following Conway's law: "Attend to Conway's Law: Have a good, modular design and use it as the basis for assigning work to different sites. The more cleanly separated the modules, the more likely the organization can successfully develop them at different sites." From the architectural viewpoint, the separation of modules has been identified as key for independent development work already as far back as the 1970s by Parnas [35].

There have been several systematic literature reviews in the area of GSD in general, as revealed by the tertiary study by Verner et al. [36]. Based on this study, it can clearly be seen that organizational factors, software engineering, the software development process, and software project management issues are the most studied areas in GSD. Notably, from the listed 24 SLR studies, only one involving software architecture design is listed. This is a review concentrating on architectural knowledge management (AKM) issues by Ali et al. [17], where they captured key concepts of AKM in GSD, to include architecture knowledge coordination practices and the most crucial challenges. Based on a meta-analysis of the literature, they presented a meta-model for AKM in a GSD environment. Several practical design related issues were found, but the focus of the study is knowledge management, rather than the more technical process of designing the software architecture, which is the focus of our research. What the meta-analysis does reflect is a clear delineation between architectural management in a co-located setting compared to a distributed development setting.

Besides the study of Ali et al. [17], several studies consider software construction and configuration [18], but they take a process viewpoint. This strongly suggests that there is a gap in architecture design related research within GSD. This mismatch between industry needs and research conducted was further identified in an evaluation of 10 years of research and industry collaboration in

Global Software Engineering [37]. Christof Ebert and colleagues listed Architecture and Design as the least researched area with only 6 out of 260 papers covering the topic over 10 years.

2.2. Concern Framework for Architecting in Global Software Development - An Overview

In 2018 we conducted a systematic literature review (SLR) on software architecting challenges and practices in GSD [19]. The SLR synthesis enabled us to construct a conceptual model, the Concern Framework for Architecting in Global Software Development. From hereon we will refer to this as the "Concern Framework". The Concern Framework is presented in Figure 1, where the challenges and practices are grouped under themes. Relationships between themes are also shown. Themes (concepts) are presented as classes; practices and challenges are given (in condensed form) as class members (coded with SLR-P1 – SLR-P9 for practices and SLR-C1 – SLR-C9 for challenges). We use the directed labeled association to mark the cases where the concepts have indisputable relationships. We use the directed dependency notation where the relationship between concepts is clear but the affect one action has on another will be context specific and vary from case to case, and project to project. Finally, inheritance is used to denote a special relationship between themes and directly derived subthemes. Additionally, two core concepts of architecting (*Design Decisions* and *Project Management*) are notated with stereotypes to distinguish under which core concept the theme falls. Overlapping concepts across classes are marked with a special stereotype "Design Decisions and Project Management".

As shown in Figure 1, practices and challenges are related to the following themes: Organization (Structure and Resources), Ways of Working (AKM, Change Management and Quality Management), Design Practices, Modular- ity and Task Allocation. While most challenges have corresponding practices, there are no practices for Change Management, Quality Management and Task Allocation. As these themes contain tough challenges that need to be addressed, by interviewing practitioners, we aim to (a) fill these gaps by identifying how these

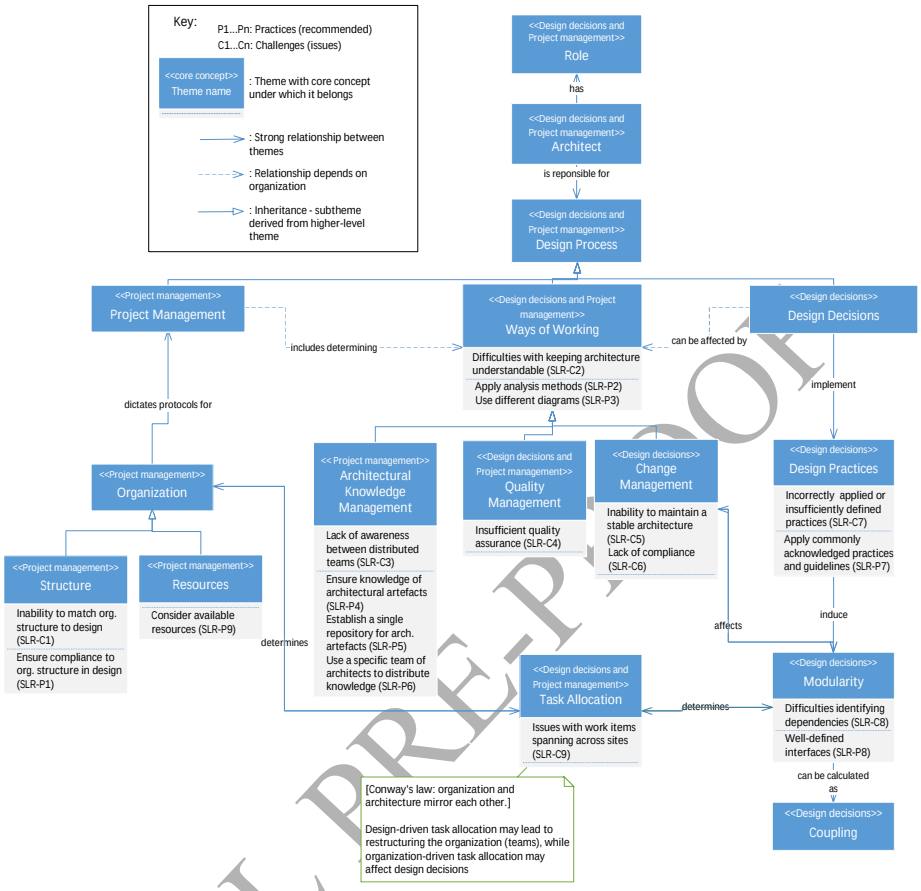

Figure 1: Concern Framework Model for Architecting in GSD [19]

challenges are dealt with in practice (b) enhance previously-identified practices and (c) identify challenges not previously identified in the literature.

The empirical study presented in this paper sets out to strengthen our findings and find answers to the following Research Questions:

RQ1: What challenges do practitioners face when designing software architecture in GSD projects?

RQ2: What practices do software architects use to accommodate the distributed nature of development work?

3. Research Method

This section presents an overview of our qualitative research method, to include sampling of practitioners in architectural design (we call 'interviewees'), qualitative data collection and analysis methods, and validation. A detailed description of our study design is available online [38].

3.1. Research setting

To answer our research questions, we performed semi-structured interviews with 13 representative architects from seven different global companies. All representatives participated voluntarily. The interviews lasted between 1 and 2.5 hours, and were performed by the first author, who recorded the interviews and wrote notes. In this purposive sample, all interviewees were selected due to their experience of working with software architecure[1] in distributed software development projects. Some had additional experience including project leadership and management. Interviewee and company backgrounds are summarized in Tables 1 and 2. Companies are coded with letters A-G. As shown, in each of companies A, B, C and F we interviewed one individual, while in companies D, E and G we interviewed three individuals. In companies D and E the interviews were performed as a group interview, while for company G all three practitioners were interviewed separately. In companies D and E the interviewees worked in very similar projects or roles, while in company G the interviewees had much more varying roles, though all related to architecting.

3.2. Questions

The Concern Framework [19] gave us a starting point for our interviews. When constructing our questions, we ensured that the topics which were poorly addressed in the literature were covered, eliciting practical examples of their architectural practices from the interviewees.

[1]Those working with architectural issues are those involved in making design decisions, prioritizing requirements and development work accordingly, and contributing to architectural artefacts, such as documentation

Table 1: Interviewees' backgrounds

	A	B	C	D	E	F	G
Number of participants	1	1	1	3	3	1	3
Field of company	Power/ Electrical automation	Software Development	IT Consulting / Software Development	Software design	Software development	Mining/ machinery and information systems	Telecommunications
Size of company (employees)	500 000	30+	100	50-60	1000	35 000	100 000
Size of IT /Software Development section	Most activities involve sw	20+	30	90%	70%	100	30%
Number of sites per project where interviewee has worked	3	2	3	4	3-5	2-5	3-4, 2-5, 12
Number of countries	4	2	4	4	3-5	5	2-3, 2-5, 9
Countries where projects have resided	USA, IT, IN	FI, PK	NL, HR, FR, SA	FI, VN, DE, JP	IE, FI, IN, PL, RO, AR	IN, FI, SE, NL, USA	FI, CN, IN, PH, RO, PL, DE, USA, FR, PT

JOURNAL PRE-PROOF

We summarise the various steps here in four phases:

Phase 1: Background The purpose, ethical considerations and background associated with the study is described to the participant. Key terms are defined, such as "GSD" and "Software architecture design" to ensure a common understanding.

Phase 2: Demographics We collect personal information such as experience and role, and also ask about the organization size and countries involved in the projects on which the participant is working (see Table 1).

Phase 3: Exploratory Questions We ask open questions on principles, practices and guidelines that the interviewees has followed or found useful (or not) in their work with GSD in general and in software architecture design.

Phase 4: Focused Questions Here we ask specific questions on themes we found in the Concern Framework, repeating known challenges and practices, and probing for answers to those challenges without a matching set of practices. For the full set of our semi-structured questions, see our interview protocol [38].

3.3. Analysis

In order to derive themes from our qualitative data, we applied a form of thematic analysis as described in [39, 40, 41] accompanied by memoing [42, 43]. The thematic analysis involved an abstraction of codes from the transcripts (termed 'codes'), which in the cases of "practice" and "challenge" were predetermined, but other codes were generated inductively from the material.

The analysis and validation process is outlined in Figure 2 and proceeded as follows:

1. **Code** each quote.
2. Create a **memo** item for each quote.
3. Extract concern.
    - Select practice and challenge coded quotes (subset of item 1.)
    - Reword long quotes into a shorter format
    - Synthesize practice/challenge codes to create a theme

- Re-iterate synthesis process

4. **Validate** by conducting an inter-rater reliability test of each code and theme, as components of the framework (involving 3 researchers).
5. **Revise** framework based on validation results (and repeat inter-rater test to check assumptions)
6. **Augment** the Concern Framework with concerns found in this study
7. **Derive GAP Framework** The new GAP framework comprises practices, challenges, concerns, and relationships, merged with our Concern Framework (see Fig 2).

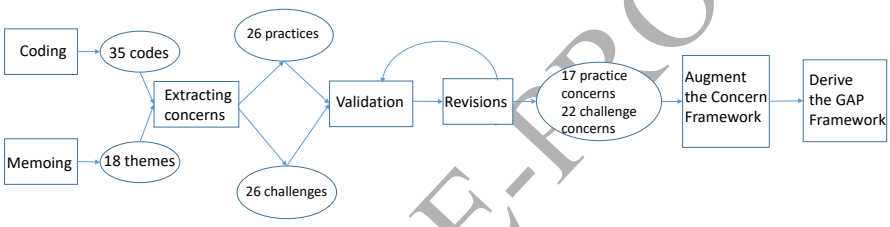

Figure 2: Analysis and validation process leading to development of new GSD Architectural Practice (GAP) Framework

A more detailed description of steps 1–5 are described in [38]. Findings stemming directly from our empirical study are discussed in Section 4.

Combining our new empirical findings with the previously derived Concern Framework creates a more complete view of architecting in GSD, which we present as the GSD Architectural Practice Framework (hereafter called the GAP Framework). We present the GAP Framework in detail in Section 5.

4. Architecting in Distributed Software Development Projects

4.1. General Views on GSD and Software Development Practices

We began our interviews by enquiring about how distributed development is carried out in the companies. To understand the operating environment dictating architecting practices, we asked a number of background questions, the answers to which are summarised in Table 2.

Table 2: Context for Distributed Architecting

	A	B	C	D	E	F	G
Number of participants	1	1	1	3	3	1	3
Common language	No shared language	English used	English used	English mainly used, problems with Asian countries	English used	English used	English used
Effect of time difference	Issues found, requires good management	Not significant, 4-5 hours	Time difference to USA requires management	Problem of varying level between sites	Some problems are experienced	3 hours time difference, not a problem	Serious problems / Not a big problem / Does have an effect, requires flexibility
Reasons for GSD	Cost	Expertise, cost	Talent, cost	Company ideology	Cost, access to people	Cost	Cost and field of business
Used software process	Agile methods	Scrum-like	Scrum	Scrum derivative	Scrum	Scrum-like	Scrum-like
Is distribution considered in the process?	Yes, sub-areas per site	Yes, in maintenance responsibility and communication	Yes in meeting arrangements, otherwise no	Yes, basic assumption	Yes, resources are considered	Partially yes	Yes in development tasks / communication
Perceived effort for architecting	Several years	25% of development time	15-20% of development time	1 full-time architect	Full-time architect team	A lot of effort, internal competition	Not enough time, or time spent but allocated to wrong things
Biggest challenge in GSD [a]	How to establish trust within and between teams	COMM, CULT	TIME	COMM, CULT, instability	COMM, TIME, CULT, dealing with hidden organization and varying structures	TIME, COMM	CULT, COMM, How to handle lost information and organize joint meetings

[a]COMM: How to handle communication, CULT: How to handle cultural differences, TIME: How to handle different time zones

We explored experiences based on different temporal distances between sites. In company B the time difference of 4-5 hours was not considered to be a problem. However, with company E, all interviewees agreed that there were problems, even though time difference between some sites was less (2 hours) or about the same (5 hours), as in company B. Most interestingly, in company G different interviewees had varying views on the effect of time differences. While G1 did not work with more or different sites than G2, he had experienced severe difficulties, while G2 did not consider any real problems. Further, G3 was working with the most number of sites, with expectedly the biggest time zone challenges, and the problem did not seem significant.

As expected, the dominant reason for distributing development is to save costs. However, the second biggest reason for the distribution is access to resources. In some cases this appeared to be acquisition of resources at a specific location; in others the companies had acquired a smaller local company to gain access to a required resource.

We note that all the companies are using, or at least are attempting to use, some variant of Scrum. The level of how strictly Scrum is applied varies, and in some cases there were distinct elements of the waterfall process still apparent.

Consideration of software development distribution varies significantly among organizations. In some cases there are clear implications that the architecture design process makes allowances for distribution of development and maintenance, but in other cases only practical arrangements with regard to communication and meetings are considered.

4.2. Role of Architect in a Distributed Environment

We proceeded by asking about the role of an architect in the companies and how architecture design fits into the development processes. The answers are presented in Table 3. Architecting work is handled quite differently across the participating companies. Several companies have a practice where a multi-site architect team or even several teams lead the work, with the architect integrated into development teams to involve them in the daily work and to

ensure architectural knowledge distribution to all developers. However, the other extreme is that there is one chief architect or a CTO having the final say on decisions. We observe that cases with one chief architect are quite different: Company D is extremely distributed (4 main office sites and a number of experts around the world), while companies B and F have the least number of sites (only 2 active sites currently) and the lowest number of different teams involved in development.

There is near consensus relating to the responsibilities of an architect - so the role appears to be the same regardless of company size and field of business. The software architect is expected to be the person who combines different stakeholders' concerns and manages design decisions at large. However, quite radical differences are found particularly within company G, where G1 considers that the architect's responsibility is to maintain interface documentation, while G3 views the architect as a negotiator. This would imply that in large organizations where there might be architecting at various levels, for example, feature, component and product line, the experience of an architect's role and responsibilities is more context specific.

Two main practices emerged on how architecture design fits with the (varying) Agile methodologies followed. One option is to allow the architecture design to evolve as development progresses. In this case, architectural tasks are considered in a similar way to other development tasks in the Scrum framework. The other option is to have a "sprint zero", where the main portion of the architecture is designed before development actually starts. This is often required by the customer.

4.3. Software Architecture Design Challenges and Practices in The Field

We asked interviewees what they considered to be the *biggest challenge* and the *most important practices* when conducting software architecting in global development projects. The following sections summarise the responses.

Table 3: Architect's role

	A	B	C	D	E	F	G
Number of participants	1	1	1	3	3	1	3
Who is in charge of decisions	Architect team. Members from all sites.	CTO, who leads a decentralized team of architects, each in charge of one sub-area.	Two central architects in charge of whole product, one architect representative in each developer team.	One Chief architect for each project	Multi-site architect teams, and individual architects from the teams. One team per sub-area.	One chief architect	Architect team, one representative from each site, different level architects, chief architect for each release.
What are the responsibilities of an architect	N/A	Responsible for the design of their own sub-areas, finding out what should be done, making a preliminary plan on how should be done.	Architectural decisions, weighing the balance of trade-offs, different stakeholder requirements. Safeguarding the implementation, communication.	Chief architect is responsible for the big picture	Linkage between company goals and how software is developed. / Doing research on options. Estimating risks. Reversability	Chop the product into the right kind of divisions, clear and reusable parts.	Preparing interface documents and ensuring functionality. Keeping people informed, enable people to make the right decisions.
How is architecture design fitted to Agile processes	N/A	Architecture designed before project starts.	Treated as normal development work, architectural tasks are tickets for the PO to prioritize and distribute.	Project started with a month long design period, high-level architecture and prototype done.	Architecture deliverables for each sprint, some design before development. Product is agile, agile architecting under discussion.	Plan is to have architecture planned 2-3 sprints ahead of time.Doesn't always happen.	Draft, specify, repair. Design interwoven in iterative work. Scrum not organization-wide. Higher level architecture not designed iteratively.

4.3.1. Challenges

Our data synthesis of participants' responses identified seven recurring themes.

Deviating from processes

Our interviewees brought up very strongly the problem of deviating from processes. They found that even Agile processes (which were used in some way in all the interviewed companies) were sometimes too strict for daily development work. This may well be a result of conflict caused by an increased need for coordination in distributed processes, while, when using Agile processes, teams are intended to be self-organizing. For example, developers in teams feel that not every small detail needs to go through the defined hoops. This becomes a problem when developers start to increasingly ignore the defined processes, ultimately leading to difficulties in task synchronization and mismatch in code and design.

This issue was not reported in the literature, but various examples from our interviewees stress the challenges it brings in practice. Processes are essential in controlling a distributed project, and deviating from them brings uncertainty, distrust, misunderstandings, delays in schedule and sub-quality software.

Handling instability

Our interviewees repeatedly raised the issue of frequent personnel and team structure changes and how it makes architecture design that much more difficult. In our SLR, we found instability to be a challenge as well, but from the point of view of changes in the architecture - in the literature, a more common problem was that the architecture was not compliant with the requirements it was supposed to fulfill as a result of uncontrolled changes to the software design. Interviewees did not find this to be an issue at all, but rather they struggled with keeping the architecture aligned with an ever-changing organization and also keeping communication channels up-to-date.

Adding to the challenge of keeping the architecture compliant with organization is the organization's proneness to instability, which is particularly em-

phasized in distributed software development. Instability manifests itself as changing team structures, changing responsibilities between sites, changes in personnel and in roles of existing personnel. Personnel changes easily lead to poor communication, as relevant communication is not reaching the correct targets anymore, and key people are missing out on information that they should be receiving.

Difficulties due to distances

Communication is well-known to be challenged by distance. Practical work suffers when communication is delayed, there is insufficient technology for web meetings, and when there are mismatches in how certain terms are understood between sites. The latter was highlighted by one our interviewees: *"But of course, there are misunderstandings all the time. That a software is ready and working means such different things in Asia and Finland."*. While communication difficulties due to distance are already well-recognized in the literature, our interviews highlighted some less-known issues: having the same software and hardware versions available and being aware of available human resources and skills.

Challenges supporting the Concern Framework

Keeping architecture compliant with organization structure. Software architecture following the organization structure resonates with Conway's law, although Conway suggested that this natural tendency might not be optimal. Perhaps given the distances in GSD, this mirroring is less obvious, and needs intervention. This challenge is illustrated with a quote from an interviewee *"Structure, structures as well. Its management structures sometimes, and, you know, people you are working with, they are working on the same piece of software or same product, but [..] they are reporting to the different editors"*

Understanding architectural decisions. An interviewee discussed conflicting assumptions: *"the geographical distance comes into play in that there are terribly many things that are not said aloud, that people assume differently in*

different countries and places, in relation to practices and all that, so those are difficult to detect. Especially if you don't meet in person, then they don't really come to light." In GSD, problems in communication and practical work easily lead to difficulties in understanding architectural decisions. This is evident in two ways: people can have conflicting assumptions about the software, or disagree about the choices behind the architecture of the software being developed. In extreme cases, this lack of transparency means that the problem only comes to light after the conflict has caused an error.

Achieving modularity and separation of concerns. The effects of disagreement are identified in another example: *"simple things like the separation of concerns, that you have the UI separately and that we don't go making anything within the UI that is clearly on the logic side, and these kinds of general practices. [...]but the problem has been that you have to keep an almost daily watch on things, that it feels like they sort of see the issue very differently in India."*. Here the architect who was interviewed reflects on a situation where an offshore team had been repeatedly told to conform to a given design and had kept deviating from it, resulting in sub-optimal software. This kind of experience shows how arguing over the architecture design can bring about serious problems and further emphasize the difficulties on separating concerns and achieving modularity.

Lacking knowledge management practices. A mismatch between how one site provides documentation and the kind of documentation another site expects may result in delays, misunderstandings and even errors in code, as mentioned by an interviewee: *"What is most certainly an issue, in the matter of intense debate, its the level of definition that should be provided by architecture."*. Understanding architectural decisions can be aided by distributing knowledge on architectural artefacts across sites. However, in GSD, sharing artefacts is not enough, as issues arise not just from lack of access, but also from a lack of knowledge as to what needs to be shared. This issue is most notable in documentation. Different sites may have very different levels of education, and are accustomed to different notations and detail given in the documentation.

4.3.2. Practices

Our interviewees found the question "What is the most important architecting practice you apply when engaged in GSD?" quite difficult to answer. Their initial answers tended towards communication issues and knowledge management. When probed and encouraged to dig deeper and think about how to solve problems, they often came back to the question at the end of the interview. Eventually we were able to elicit ten concrete design practices, four recommendations regarding task allocation, and three notes on general practices.

Consider existing product and its constraints

Software is often built on top of existing software or hardware, which presents limitations. Open source components and libraries cannot be chosen simply for the needs of the new extension but need to be checked for compliance with the existing product. Further, in order to achieve modular software as a whole, dependencies within the existing product and between existing and new code must be considered particularly carefully to aid the distribution of development work.

Apply continuous integration.

Utilizing the continuous integration pipeline will aid in showing flaws quicker and open the codebase for all sites. Many synchronization issues are eased and low-level quality issues are handled with automated testing. Essentially, continuous integration was found to solve problems related to one site working on a piece of code, and other sites just waiting to receive a completed block to even begin their own work.

Create product boundaries based on Application Programming Interfaces (APIs).

APIs are a widely-recognized practice, and are a well-specified and widely-spread way of handling interfaces and boundaries between modules. However, our interviewees also emphasized their use in the context of product boundaries.

Consider maintenance responsibilities as a driver for task division.

In practice, we found that sub-optimal task division during development time was well-compensated by a more optimal task allocation during the maintenance phase. In fact, maintenance is optimally done by the same team who created the original code, and maintenance often spans a longer time-period and more changes than the initial creation. This clearly deviates from recommendations found in the literature, where maintenance is often not considered at all when discussing design time activities in this context. Allocating tasks to those who end up doing the maintenance work can be optimal in the long run, even though, during development time, the division would be sub-optimal regarding schedule or expertise.

Practices supporting the Concern Framework

Determine driving architecture style. Interviewees stated that the driving architecture style was not always clearly defined, but only assumed, resulting in conflicting assumptions. However, the chosen style is a driver for all subsequent decisions. Starting architecture design from determining a driving architecture style is a basic concept. In practice, when people are contributing across the globe and communication is difficult, a consensus on what the architecture style is or whether a decision has been made may actually be missing.

Determine platform to base design on. The chosen platform will limit subsequent design choices regarding utilized technologies, as compliance must be considered. Again, while such an action should be done at the very start of the design process, ambiguities easily exist in a distributed environment. This, as well as the previous recommendation, quite naturally falls under "Apply common architecting practices" that was listed as a key practice in the Concern Framework.

Create microservices to separate development items. A distributed project aims for distributed development items, and microservices were considered a particularly suitable paradigm. This is quite clearly a specification of "determining an architecture style", and resonates with the recommendation

of using the Service-Oriented Architecture approach as found in the Concern Framework.

Create a proof of concept and **Create demonstrations.** A demonstration shows potential problems better than documentation. A proof-of-concept, in turn, aids demonstration between sites. These recommendations resonate with the practice of creating prototypes that is present in the Concern Framework.

Base task division on layers. Interviewees found layers to be the clearest separation of tasks. This particularly this applies to cases where the layered architecture is used.

Task allocation

The following three recommendations all convey the same message - separation of development tasks between sites - from slightly different viewpoints. This ideology could be considered to contradict the recommendation of using continuous integration that opens the codebase for all. All these recommendations are in line with practices found in the literature, encouraging an architecture-driven work allocation and retaining tightly coupled items on one site.

Keep development of core product at one site. As key business is based on the core product, it was considered important to keep quality high by not distributing the core development.

Clearly separate responsibilities between different sites. This helps coordination, control and keeping the design intact.

Avoid leakage of site-specific functionalities between sites. Site-specific functionalities should be tightly kept at the assigned sites to ensure quality.

General views

Finally, there are three general views regarding the architecture design process, all strongly supporting the views found in the literature:

- Establish practices to enhance knowledge distribution across sites.

- Have clear roles to aid in governance.
- Align architecture and organization

Interviewees found that engaging with and involving developers in the decision increased their understanding of architecture and commitment, for example - *"the team participates in the architecture work so that its a way to get the team to commit, them taking part in the planning of the architecture."*. They also found direct benefits from using Agile methods particularly in the distributed context. For example, daily or weekly Scrums increased communication, which in turn led to fewer incorrect assumptions. To truly facilitate distributed development, having mechanisms in place that enable knowledge distribution is a first step, but a necessary second step is to create a working culture where the need for increased communication between sites is recognized and possibly enforced. The keyword here is thus, *enhancing*. One mechanism to accomplish this is to engage developers from all sites into the architectural design process.

All interviewees confirmed that their teams applied a form of Scrum methodology, where the teams are given a level of autonomy to self-organize. Thus even architectural work would be the responsibility of the teams. However, interviewees strongly supported having someone external to the teams to make the architectural decisions in the GSD context, particularly due to dependencies between sites that teams may not be aware of. Further detailing the architect's role, they advise that architects handle all relevant communication between different stakeholders. There should be a clearly named person in charge of managing knowledge distribution, architectural work and prioritization.

Finally, our analysis of the interview data partially supports Conway's law, as interviewees highlight how the organizational structure guided the design of the software architecture. However, two opposing alignments were observed: (a) in line with Conway's law, the organization acts as a driver, and the architecture design is based on skills, resources and the communication structure in the organization and (b) - the opposing view, the architecture acts as a driver, with resources moved and acquired based on the needs of the architecture. One

interviewee when asked, whether the architecture drives the organization or the organization drives the architecture, stated: *"It's an evolution"*.

5. GSD Architectural Practice (GAP) Framework

This section demonstrates how we take the results presented in the previous section, and combine them with our Concern Framework (presented in section 2.2) to create the GAP Framework shown in Fig 3.

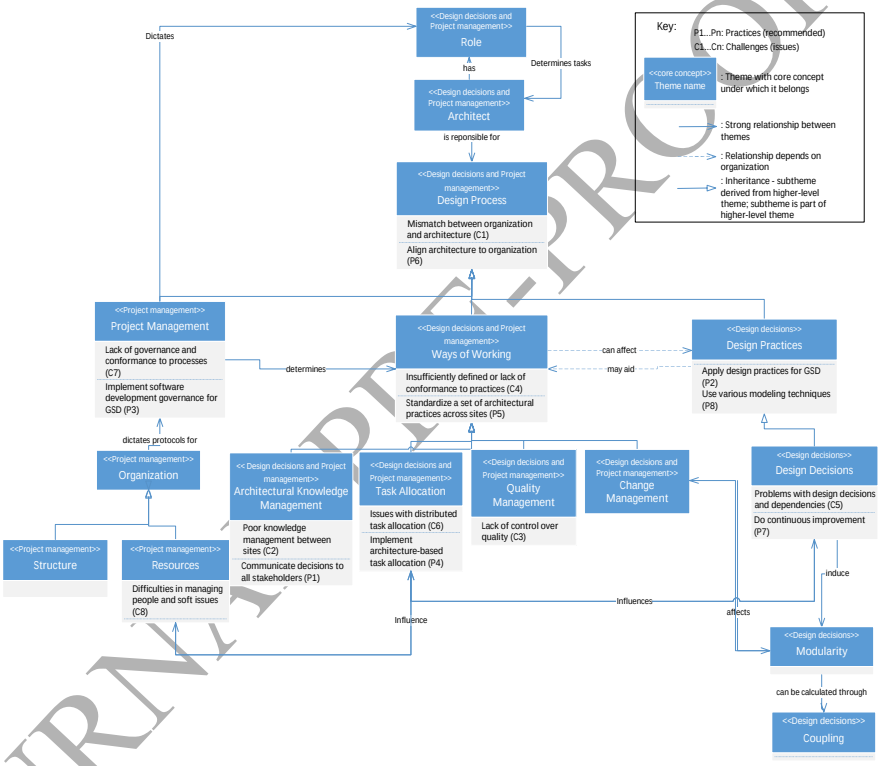

Figure 3: The GAP Framework

5.1. Conceptual Model

Each challenge is given the ID tag "C" with a running number, so each challenge has a unique ID number. Similarly, each practice is given the ID tag "P" with a running number, so each practice has a unique ID number. Practices

that are under the same theme as a corresponding challenge are natural solutions to that challenge. However, practices that are associated with challenges via relationships can also be helpful. The complete mapping of practices to challenges is given in Table 4 with the interpretation of relationships illustrated in Figure 4.

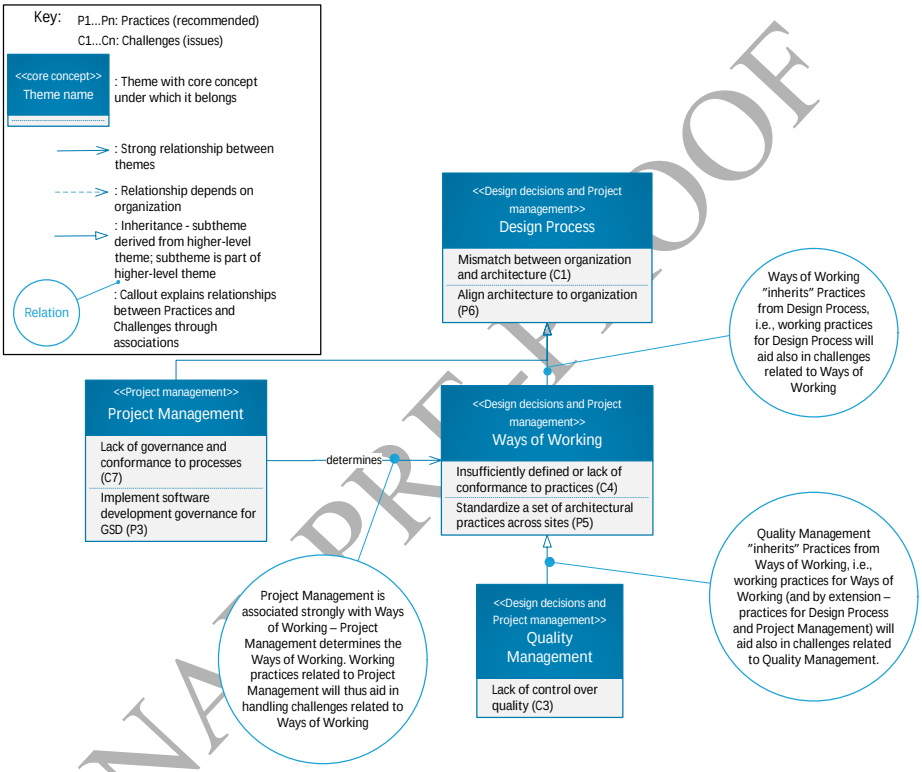

Figure 4: Illustration of relationships

The evolution from the Concern Framework to the GAP framework is summarized as: New relationships between Project Management and Role (of Architect), and Role and Architect were added; Task Allocation was placed as a sub-theme under Ways of Working; Relationships between Task Allocation and other concepts were modified; The relationship between Design Practices and Design Decisions was modified so that Design Decisions are now part of Design

Table 4: Mapping of Practices to Challenges

	C1	C2	C3	C4	C5	C6	C7	C8
P1		x						
P2	x			x	x			
P3		x	x	x			x	
P4						x		
P5			x	x	x			
P6	x		x					
P7					x			
P8		x			x			

Practices; The relationship between Project Management and Ways of Working was fortified to be a clear association instead of depending on the Organization; The relationship between Ways of Working and Design Practices now works in both directions.

Identification of increased dependencies on the architect's role and how task allocation fits into the model was significant in our empirical study. We found that the role of architect in GSD is dictated by project management practices. Organizing architecting work to one chief architect, to architects on several levels or to a team of architects who may be also involved with development, has a large impact on what architecting means in each particular case. Depending on the role, an architect may be involved in practical work regarding architectural decisions and participate in implementing them, or act more as a mediator between stakeholders and lower-level architects. Task allocation, in turn, was found to be part of Ways of Working, defined by Project Management practices. Ways of Working (and by extension, Task Allocation), may influence Design Practices. This depends on the state of evolution of the organization and the architecture. As in the previous model, Task Allocation influences on Resources and Design Decisions, and vice versa.

5.2. Tackling Challenges

Elicited practices and challenges with their related concerns are given in Tables 5 – 11. The concerns related to each Practice and Challenge are labeled

with the corresponding ID, followed by "co" (as in concern), and a running number. Additionally, each concern is given a postfix of "slr" if it was derived in our SLR or "emp" if it was a result of the empirical study presented in this paper. Challenges are presented via themes found in the conceptual model, and we will discuss how they can be alleviated via the associated Practices. In the tables, we present those Practices that are placed under the same thematic concept as the Challenge(s) in question. Please note, though, that as illustrated by Figures 3 and 4 and Table 4, that Practices under different thematic concepts can also aid in answering Challenges.

5.2.1. Design Process and Considering Quality

We combine Challenges for Design Process and Quality Management, as the Practice for Design Process is the one most closely linked to Quality Management.

During the Design Process the architect should carefully consider matching the architecture with organizational structure (C1), as this will significantly aid in further decisions and particularly task allocation. Because they are working in a distributed environment, an additional aspect to this challenge, is that organizations often have an unstable structure. The concerns brought forward by the interviewees (C1_co2_emp, C1_co3_emp) are very similar to those already found in the literature – matching the architecture with the organization structure is difficult.

The Quality Management related challenge (C3) highlights the need for proper quality assurance, with new concerns brought to light by practitioners. While interviewees mention the importance and benefits of arranging architectural reviews and having good testing coverage in the distributed setting, they are more difficult to arrange this context (C3_co4_emp). For example, interviewees reported concerns regarding insufficient recording of quality requirements (C3_co6_emp). Additionally, different sites may have different aptitudes for running automated tests (C3_co5_emp). These concerns are also addressed as part of P6, which raises quality management practices as a separate concern

Table 5: Design Process and Quality Management

ID	Challenge/Practice	Concerns
C1	**Challenge**: Mismatch between organizational structure and architectural design and difficulties in dealing with instability	Lack of alignment between architectural decisions to organization structure and not reflecting architectural changes to organization (C1_co1_slr) Challenges brought by misalignment between organization and architecture (C1_co2_emp) Challenges brought by personnel changes (C1_co3_emp) Difficulties ensuring compliance of modular design throughout the lifecycle and changes in organization (C1_co4_slr) Inability to retain experts from all domains required for change implementation (C1_co5_slr)
C3	**Challenge**: Lack of control over software quality	Delegating design decisions to local team deteriorates quality (C3_co1_slr) Insufficient quality management (C3_co2_slr) Decentralized data and state management lead to inferior quality (C3_co3_slr) Insufficient methods for reviewing architecture design against quality demands (C3_co4_emp) Insufficient automation for testing, a lot of manual tests (C3_co5_emp) Insufficient recording of quality requirements. (C3_co6_emp)
P6	**Practice**: Align architecture with organization arrangement	Include business goals in design (P6_co1_slr) Base architectural decisions on available resources (P6_co2_emp) Establish quality management practices (P6_co3_emp)

(P6_co3_emp) when aligning architecture and organization.

We recommend aligning architecture with organizational arrangement (P6) – the processes, practices and resources – in addition to purely aligning it with the organizational structure. Our interviewees particularly highlight the need to base decisions on available resources (P6_co2_emp) – here resources includes the effort developers can put into their work, developer skills and technology

experience, location of team members, access to hardware, and software licenses. However, as demonstrated, changes in personnel (C1_co3_emp) will easily break this alignment, and thus the architecture should be flexible enough not to depend on individuals with the potential of creating bottlenecks.

Design Process combines Project Management and the actual Design Decisions. Thus, while well-managed Practices from above will reflect well also on lower-level concepts (as illustrated in our conceptual model in Figure 3 and the relationships in Figure 4), in this case Design Process will benefit when the parts making up this high-level concept are in order. Concerns related to Design Practices as detailed in P2 (Table 8) will further aid in aligning organization and architecture, and concerns related to P3 (Table 10) and P5 (Table 7) will help improve quality.

5.2.2. Handling Architectural Knowledge Management

Architectural knowledge management (AKM) is a major challenge, as distance makes traditional communication difficult or even impossible. Demonstrated in many ways, deficient AKM (C2) is quite often experienced by interviewees. Proper knowledge management entails ensuring that all sites have access to documentation and that such documentation is understood (highlighted by concerns C2_co1 – C2_co6). There are often various documentation repositories, wikis, and tools where documentation is added. However, in a distributed setting it easily becomes unclear who has access to these systems, who accesses them, and when someone does access the documents, whether the system is structured so that documents can be found when needed. Further, when projects are distributed, and thus project management is also distributed, communication across project boundaries becomes more challenging (C2_co11_emp).

In modern software development it is common to rely on shared libraries and components. Thus, when the maintenance responsibilities of such components are distributed across a variety of projects, and management of those projects, in turn, is distributed across the globe, there is an increased threat that shared libraries are not kept up to date or their ownership becomes unclear, leading to

Table 6: Architectural Knowledge Management

ID	Challenge/ Practice	Concerns
C2	**Challenge**: Poor architectural knowledge management between sites	Difficulties in effective creation and sharing of architectural artifacts (C2_co1_slr) Difficulties in maintaining a common view of the project (C2_co2_slr) Inconsistent usage of electronic systems for knowledge sharing due to preference of social networks (C2_co3_slr) Insufficient architectural documentation (C2_co4_slr) Insufficient documentation practices (C2_co5_emp) Insufficient knowledge management practices between projects and across organization (C2_co6_emp) Disagreement in design choices (C2_co7_emp) Problems recognizing and caused by conflicting assumptions on software (C2_co8_emp) Insufficient understanding of architectural decisions in teams and other stakeholder groups (C2 co9_slr) Incorrect assumptions made during design (C2_co10_slr) Communication issues due to distances (C2_co11_emp) Unclear ownership of architectural elements (C2_co12_slr)
P1	**Practice**: Communicate architectural decisions to all stakeholders	Establish practices enhancing communication and knowledge distribution (P1_co1_emp) Architects should handle communication with different stakeholders, considering stakeholders' background (P1_co2_emp) Communicate architectural artefacts and practices clearly to all sites (P1_co3_slr) Arrange collocated activities for architecture team to promote awareness (P1_co4_slr) Establish a team of architects for handling communication between different stakeholders and teams (P1_co5_slr) Ensure understandable and accessible documentation for all parties (P1_co6_emp) Maintain a single repository for architectural artefacts accessible to all (P1_co7_slr)

a variety of problems when developers unnecessarily attempt to duplicate their functionality (C2_co12_slr).

Our empirical study draws attention to disagreement in design choices (C2_co7_emp), which closely relates to insufficient understanding or incorrect assumptions on said choices (C2_co8_emp, C2_co9_slr, C2_co10_slr). While disagreeing and raising issues about potential drawbacks of certain choices is a natural part of architecting, the concern that was specifically highlighted in the distributed setting arose due to difficulties in communication and not having enough access to knowledge. When there are limited possibilities for developers at remote sites to attend meetings and discuss the design with the architect, they are less likely to understand all the constraints and drivers behind the decisions, and thus, they end up questioning the selected solutions.

These challenges can be alleviated to some extent if architectural decisions are communicated to all stakeholders (P1) – a practice that experienced architects are no doubt aware of. However, our detailed concerns presented may help architects notice gaps in how communication is handled. It is not enough to simply put information out there, but those responsible for communication (P1_co5_slr) should also consider the stakeholders' background and adjust their method of communication accordingly (P1_co2_emp), ensuring that documentation is not just available, but also understandable and accessible (P1_co2_emp). In general, communication practices should not just exist to allow communication, but should be designed in a way that enhances communication (P1_co1_emp). This can include visiting remote sites and having common fixed meetings.

Practices related to software development governance (P3, see Table 10) may also aid in improving knowledge management. For example, we recommend having a representative architect on each site and engaging developers in architectural work. Further, utilizing various modeling techniques as detailed by P8 (see Table 8) may improve knowledge management via an increased level of understanding, as stakeholders with different backgrounds may find some diagrams more usable than others.

Table 7: Shared Practices

ID	Challenge/ Practice	Concerns
C4	**Challenge**: Insufficiently defined or lack of conformance to shared practices across sites	Inconsistent versioning (C4_co1_slr) Insufficient interface specifications (C4_co2_slr) Ignorance of or incorrect use of principles, rules and guidelines for architectural design and knowledge management (C4_co3_slr) Lack of stability in architecture leads to difficulties in applying design rules and dividing tasks (C4_co4_slr) A lack of conformance to architectural specification (C4_co5_slr)
P5	**Practice**: Standardize a set of architectural practices across locations	Ensure that teams develop code based on common design agreements (P5_co1_slr) Use common architectural practices and ensure they are well-defined (P5_co2_slr) Consider a service oriented approach (P5_co3_slr) Take advantage of Agile methods (P5_co4_emp) Use prototyping (P5_co5_slr) Ensure fit to requirements (P5_co6_emp)

5.2.3. Ways of Working

How to do and what kind of practices are established in design process and development are defined in Ways of Working. In the GAP Framework we present concerns related to insufficiently defined practices or how practices were followed across sites (C4), which can be solved by using standardized set of practices across sites (P5). Therefore, all those involved in architecting work should have a common agreement on what particular practices and drivers are applied in design (P5_co1_slr). This is not a given in distributed projects. Furthermore, our current study identified further practices to alleviate this concern, for example, (P5_co6_emp). Architecture design stems from eliciting functional and non-functional requirements, and creating the architecture to reflects these needs. However, if the design work is not well-coordinated, the original requirements may fade into the background, resulting in compliance issues in the long run, especially in a distributed setting (C4_co5_slr). This may be aided by utilizing

Agile methods (P5_co4_emp) - handling a smaller set of requirements (or user stories) at a given time. This allows the architect to quickly adjust development work in an unstable organization, and thus will aid handling compliance and communication issues. It can also help to discover misunderstandings in a more timely manner.

Ways of Working can be further improved by using solid design practices particularly suitable for GSD (as detailed in P2, see Table 8), and by implementing software development governance (P3, see Table 10), which is essential for Project Management, which in turn largely defines Ways of Working.

5.2.4. Architectural Design Decisions

When architectural design is itself distributed or needs to consider distribution of subsequent development work, challenges identified relate to reaching viable decisions and handling dependencies (C5). In addition to the most common concern of insufficient decoupling, as strongly stressed in the literature (C5_co1_slr), interviewees note how the complexity of the product brings challenges to the architecture design (C5_co2_emp) regardless how the project is organized. However, complexity is an even bigger risk if architecture work is spread over several sites, and a distributed team needs to gain a common understanding of the solutions and choices to deal with the complexity.

While modularity and coupling were already identified as key concerns in the Concern Framework (C5_co1_slr, C5_co5_slr), in our empirical study such concerns were complemented by challenges faced by the interviewees: finding entities in the architecture between which interfaces can be designed (C5_co3_emp), and understanding and eliminating dependencies (C5_co4_emp). Modularity is as big a concern in collocated projects as it is in distributed projects, but as task allocation is critical for the success of distributed projects, and that, in turn, is highly dependent on the modularity of the architecture, concerns related to modularity should be highlighted.

To address these challenges, we found several practical concerns related to modularity and separation of concerns in the architecture (P2_co2_emp and

Table 8: Architectural Design Decisions

ID	Challenge/ Practice	Concerns
C5	**Challenge**: Problems associated with architectural design decision and identifying dependencies	Insufficient decoupling, cross-component features (C5_co1_slr) Challenges brought by the complexity of software (C5_co2_emp) Difficulties defining logical entities and finding interface boundaries in architecture (C5_co3_emp) Insufficient or no methods for identifying, understanding or preventing dependencies (between decisions, components or other software artefacts) within architecture (C5_co4_emp) Inability to recognize dependencies between or created by architectural decisions. (C5_co5_slr) Lack of time and schedule pressures affect architectural decisions (C5_co6_emp) A lack of compliance to the business process (C5_co7_slr)
P2	**Practice**: Apply architectural design practices for global software development	Implement well-defined interfaces to increase modularization and aid loose coupling (P2_co1_slr) Make interface design a priority (P2_co2_emp) Ensure components that will be dispersed to distributed teams are loosely coupled or otherwise plan component breakdown to independent modules based on distribution of teams (P2_co3_slr) Strive for high modularity and separation of concerns (P2_co4_emp) Locate dependencies within architecture (P2_co5_emp)
P7	**Practice**: Do continuous improvement	Do active research on new technologies and practices (P7_co1_emp) Consider long-term effect of design choices (P7_co2_emp) Emphasize reuse (P7_co3_emp)
P8	**Practice**: Use various architecting modeling techniques	Use (call) graphs/matrices to depict and detect coupling (P8_co1_slr) Use visualization of decisions/metrics (P8_co2_slr) Use collaborative modeling (P8_co3_slr) Use a variety of diagrams promote awareness (P8_co4_slr) Don't over-rely on UML diagrams (P8_co5_slr)

P2_co4_emp) which are particularly relevant for the GSD context. Our interviewees particularly stressed the importance of locating dependencies within the architecture (P2_co5_emp), recommending the utilization of checklists, illustrations, tools and feature-based development. In a related practice concerning continuous improvement (P7), the interviewees also stressed the possibility of reuse (P7_co3_emp), which is also easier if the design is modular. Considering that the long-term effect of design choices (P7_co2_emp) stems from similar experiences – short-term choices may lead to difficult dependencies between technologies that will be difficult to maintain. Finally, design can be aided by utilizing various architecting modeling techniques or visualizations (P8) to help share a common understanding of the decisions. (see Table 7).

5.2.5. Task Allocation

Modular design is highly recommended for GSD, as task allocation is often based on the assumption that modules or concerns are clearly separated and decoupled. But, task allocation in a distributed setting (C6) easily becomes challenging if dependencies between tasks and subsequently between teams are not identified (C6_co6_slr). Due to communication difficulties there is often more effort and coordination required (C6_co1_slr, C6_co2_slr), while decreased visibility to remote sites and what resources are truly available may lead to a mismatch between tasks and resources (C6_co5_slr).

Additionally, while work items are, where possible, often kept separate between sites in a distributed setup, multiple sites may be developing large modules which ultimately need to fit together for the final product. If one module is delayed, integration will, in time, come to a halt (C6_co4_emp).

We recommend an architecture-based task allocation (P4) supported by the literature (P4_co1_slr, P4_co2_slr, P4_co5_slr). Interviewees further raise the issue of alignment. The architecture may act as a driver, and additional resources may be acquired to fulfill the needs of the designed architecture (P4_co3_emp). Alignment between the organization and architecture can be used to allocate tasks, ensuring that resources at a given site actually match the task given to

Table 9: Task Allocation

ID	Challenge/ Practice	Concerns
C6	**Challenge**: Issues with task allocation in a distributed setting	Increased amount of effort with modifications involving several developers across different sites (C6_co1_slr) Increased needs for coordination when using experts from different sites (C6_co2_slr) Difficulties evaluating work input due to distribution (C6_co3_emp) Difficulties in synchronizing tasks (C6_co4_emp) Insufficient matching of code to available resources (C6_co5_slr) Difficulties with correctly identifying dependencies between work units and thus assigning work to distributed teams (C6_co6_slr) Insufficient prioritization rules (C6_co7_slr)
P4	**Practice**: Implement architecture-based task allocation in global software development	Identify where the domain expertise lies and allocate tasks accordingly (P4_co1_slr) Retain tightly coupled work items at one site (P4_co2_slr) Acquire and arrange resources based on architecture (P4_co3_emp) Base work allocation on available resources and minimize need for communication between sites (P4_co4_emp) Let the architecture determine how tasks are allocated, and who is responsible for each task (P4_co5_slr)

them, and that communication between sites is minimized (P4_co4_emp).

5.2.6. Project Management

Governance is an essential part of Project Management. Thus, there are inevitable challenges if governance is lacking or processes are not being followed (C7). Lack of governance may be observed when organization management is not considered in the design process (C7_co2_slr) or in dividing tasks (C7_co1_slr). We have also identified that knowledge management problems arise due to poor governance resulting in bottlenecks (C7_co7_slr) or in lack of

Table 10: Governance and Processes

ID	Challenge/ Practice	Concerns
C7	**Challenge**: Lack of governance and compliance to processes	Difficulties with making the organization reporting structure match the geographic distribution of tasks (C7_co1_slr) Overlooking organization management (C7_co2_slr) Challenges due to inconsistent standardization, tools and equipment between sites (C7_co3_emp) Schedule is prioritized over processes (C7_co4_emp) Challenges fitting practical work to defined processes (C7_co5_emp) Problems caused due to not involving a technical architect (C7_co6_slr) Impractical condensing of knowledge due to high dependency on one lead architect (C7_co7_slr)
P3	**Practice**: Implement software development governance for global software development	Assign responsibilities for prioritization, managing architectural work and sharing knowledge to teams (P3_co1_emp) Break work items to easily manageable pieces (consider one subsystem, can be handled by one person) (P3_co2_slr) Define clear responsibilities for architecture team to handle changes that span through several components and/or sites (P3_co3_slr) Ensure each site has representative architect (P3_co4_slr) Engage developers across sites in architectural work (P3_co5_emp)

expertise in design work (C7_co6_slr). Our interviewees also noted problems related to inequality between sites (C7_co3_emp).

They further reported problems related to how processes are followed. In some cases they were not able to follow the process as defined when they would have wanted to - this happened when tight schedules dictated that shortcuts needed to be taken (C7_co4_emp). In a converse case, interviewees felt that the defined process did not match practical development work (C7_co5_emp), and

work needed to be done "under the hood" to be able to do it efficiently.

One key concern is how to engage developers across sites in architectural work (P3_co5_emp). Engaging developers from various backgrounds and sites will aid in condensing and sharing knowledge and finding expertise. Similar benefits regarding knowledge management can be achieved by appointing people and giving them clearly defined roles (P3_co1_emp).

Also note that while we did not particularly map any other Practices to C7, concerns related to the Decision Process may aid in addressing the aforementioned issues. This particularly relates to organizational aspects, as demonstrated by the relationship between Project Management and Design Process in our conceptual model (Figure 3).

However, with project management issues we note a gap in how the found practice and the related concerns address concerns raised particularly by the interviewees. We did not find particular concerns that would directly aid in issues related to processes.

5.2.7. People Management

Table 11: Managing People and Soft Issues

ID	Challenge/ Practice	Concerns
C8	**Challenge**: Difficulties in managing people and handling soft issues	Lack of commitment to software development processes and guidelines (C8_co1_emp) Lack of commitment or interest in work items (distributed across sites) (C8_co2_emp) Misaligned interests and undesirability of tasks make task distribution challenging (C8_co3_slr) Challenges in development work due to cultural differences in getting things done and reporting progress(C8_co4_emp)

Our interviewees experienced a lack of commitment in a variety of ways (C8_co1_emp, C8_co2_emp) for example, there was a lack of commitment to executing the design and reporting progress (C8_co4_emp).

While we did not find direct Practices to address this Challenge, handling

such soft issues is alleviated when concerns related to Project Management and Decision Process are well-handled, as shown in our conceptual model. In particular, P3 (Implement software development governance for GSD) contains one concern which encourages engaging developers across sites (P3_co5_emp). While this relates to governance, the reason why interviewees gave this particular recommendation is strongly linked to commitment and motivation – giving a feeling of responsibility.

6. Discussion

6.1. Architecting in GSD

The motivation for conducting the empirical study presented in this paper was to broaden our understanding of architectural design methods as applied in distributed software development. While the Concern Framework we developed [19] illustrated general problem areas and lessons learned, we were uncertain as to the completeness or consistency of our results. Conducting this follow-on study has enabled us to identify further challenges and practices from the practitioner's perspective, resulting in a holistic view as presented in the GAP Framework. A recurring theme across our group of interviewees was the difficulties they, as architects, experienced when teams deviated from the defined development process and architectural plans. This divergence in the distributed setting happened too regularly, mainly because the development process was unclear, or because the teams took a different view.

Most interviewees stated their process was "Scrum-ish" - the idea was to use Scrum, but the process did not go by the book. This hybrid approach is fairly typical according to a recent large scale study of Agile adoption in GSD [44]. While a hybrid software development process might be what is commonly used, in the case of architecture compliance across teams, a mixed and possibly vague process is causing conflicting views of the architectural design.

The recommendation is for the choice of practice to be based on a common denominator: agreement across all stakeholders. This includes agreeing

on management practices and collaboration, common design principles, roles for different tasks and making sure that the organization and architecture are aligned. When development is distributed, applying commonly agreed principles and loose coupling clearly helps, as there is less need to explain choices to remote sites, and the tasks can be more clearly separated.

Misalignment between organization structure and the software architecture is a big challenge. The environment in a distributed setting can change quickly and regularly, and can result in organizational instability. If Conway's law is being observed, the tendency is for the architecture to be based around the organizational structure. How can the architecture remain stable if this is the case? With the organization continually changing. Therefore, keeping pace with changes is particularly challenging for those responsible for the architecture. We have identified that the architecture and organization need, in this case, to continually evolve over time, but the architect is continually playing a kind of 'catch-up'.

There are similar challenges regarding communication and knowledge management. Architects need to be aware of how much these are due to differences in both working and ethnic culture. Interviewees reported the frustration they had with some practitioners hiding bad news (known as the 'mum effect'). This might be down to cultural differences, where in some cultures giving a good impression overrides flagging a problem [45]. Yet handled correctly a cultural mix can enhance development with a rich range of perspectives [6].

Further, while the use of well-defined interfaces is recommended e.g. Pereira et al. [46] and Clerc et al. [47], we have noted that there are issues with the development of well-defined interfaces in the distributed organization and finding the correct boundaries for such interfaces is sometimes very challenging.

Overall, due to the distribution of software development, we have noted new architectural design concerns that have emerged within our study. In addition, such concerns became exaggerated due to the distributed nature of software development. When tasks are distributed, it is critical for the architect to recognise these difficulties, and the GAP Framework presented will support

them in doing so.

6.2. Threats to Validity

We will consider threats to validity as described by Wohlin [48] and cover the points which are relevant to our study.

6.2.1. Conclusion Validity

Conclusion validity concerns the correctness of conclusions drawn. Searching for specific results, i.e., *fishing*, is a threat which may occur in interviews that are poorly designed, or in which participants are chosen to bias the results. The interview questions were drafted in a way that they allowed very broad and thus varied answers. We also only selected interviewees solely based on their expertise and we had no prior knowledge as to how they would consider the questions or what their attitude would be towards the topic. Finally, we need to consider the threats posed by having the GAP Framework validated by authors only. We performed our analysis so that one author produced an initial framework, and two other authors validated it by mapping quotes to themes. The validating authors were given the quotes and themes separately and independently, and no indication was given of how the first author had done her initial mapping. We required 100% agreement in mapping to proceed. While this type of approach is common and similar to content analysis, we acknowledge there is a small risk of author bias. However, our study was an exploratory one, and as we did not expect any particular results, no author was set on a specific theme, either.

To alleviate the threats related to *reliability of treatment implementation*, the same interview protocol was followed for all interviewees. The only difference was that two interviews were conducted via Skype, while others were done in person. However, with the Skype interviews video connection was also included to make it as personal as possible. Small connection problems might have affected the experience from the interviewees' viewpoint, though. These are also the only occurrences of *Random irrelevancies in experimental setting*,

which may have affected the interviewees' attitude and thus the way questions were answered.

6.2.2. Internal Validity

Internal validity threats are influences that may affect the variables with respect to causality. They can be sorted into three categories: single group threats, multiple group threats and social threats. The ones applicable to our experiment are single group threats.

There is a risk related to *maturation*, i.e., that subjects react differently as time passes. Some of the interviews took over two hours of time, and it could be seen that some interviewees were getting tired at the end of the interviews. However, we had designed the interview protocol so that the most broad and difficult questions were in the beginning, and in the end were quite straightforward and simple questions, which should alleviate this threat. The design of the interview protocol is also an *Instrumentation* related threat, and has been already discussed in relation to *Fishing*.

6.2.3. Construct Validity

Construct validity concerns how well the results are generalizable to the concept or theory behind the experiment. Threats include, e.g., *mono-method bias, inadequate preoperational explication of constructs* and *hypothesis guessing* [49]. It is natural to assume that the participants had a pre-defined view of especially the challenges we were looking for, and could perform hypothesis guessing. However, in our case, there were no "right" or "wrong" answers, and thus "correct" guessing of the hypothesis would not have benefited us in any way. Further, we could observe that the answers often would initially deal with managerial issues. To undercover practical architecting challenges and practices follow-up questions were almost always required.

6.2.4. External Validity

External validity, in turn, concerns how well the results are generalizable to industrial practice. As this study was conducted with a cross-section of

practitioners currently working in the industry, we are moving closer to being able to generalize the results to other GSD organizations. However, given the relatively small sample, we cannot be too confident that every practice we list will apply to every context. For example, even within our small sample we could see how the applicability of practices depend on the kind of system that is under design and what kind of processes have been defined.

7. Conclusions

In the study presented in this paper, we collected detailed information relating to architectural design for GSD. Through several interviews with architects (all operating in a distributed environment) we gained visibility into the kind of challenges that they encountered in their day-to-day activities. These challenges include how they design and allocate tasks across their multi-site teams. We also asked interviewees how they tried to resolve the challenges. In this way, we developed the GSD Architectural Practice Framework, augmenting our previously developed Concern Framework with more detailed context, challenges and practices [19].

The challenges for the GSD architect are manifold. While we knew about the challenges in trying to match the architecture to the organizational structure, and this was given as a recommendation, we now understand more about why this is difficult to achieve in GSD. The structure is shown to be continually changing, and is unstable. Therefore, there are suggestions that the architecture should be independent of the structure, so that all stakeholders have a clear understanding of how tasks are allocated, or that the architecture should align with the structure (through modularity). Further, our study suggests that striving for alignment, our companies actually work both in line with and against Conway's law - the organization and the architecture end up mirroring each other through an evolutionary process, where both dynamically change to adapt to the structures of the other. To successfully implement such a dynamically evolving architecture, struggling to adapt to organizational changes,

the organization needs an architect with a clear vision and a firm grasp of the original requirements.

This paper's main contribution is to elaborate the dependencies associated with the architect's role, particularly the architect's role in task allocation in a global setting. The architect does not work autonomously since design decisions are strongly influenced by project management practices. We observed that in some companies one architect is responsible for the overall design decisions, whereas in other cases it would be a group decision (with a team of architects). Although all participants applied Agile methodologies, there were pros and cons. For example, on the positive side interviewees found Scrum ceremonies supported improved communication across sites as wrongful assumptions could be detected earlier. However, in some cases the expectation that teams are self-organizing and are responsible for the day to day development, made it challenging to impose architectural decisions from outside the team - something that is often necessary when part of a larger project involving many teams and sites. Going back to handling a dynamic architecture in an unstable environment, leaving too many decisions to self-organizing teams in such an environment may very easily lead to an architecture that is no longer in compliance with requirements, if there is no clear ownership. Visibility across sites, teams and the lifespan of the product is also required to make a truly optimal task allocation and architecture plan, as one of our key results is that development of certain components are preferably allocated to those who will also be maintaining those components – if maintainability is a significant quality requirement or there is expected to be a high level of reuse of the components.

The dependencies in our newly derived GSD Architectural Practice Framework (GAP) further illustrate the complex inter-relationships of challenges to practices and the holistic nature of architectural design in GSD, where the recommendation is to consider applying these GSD architectural practices to achieve a desired balance.

8. Acknowledgements

The work of the first author was supported by the Academy of Finland. This work was partially supported (second and third author) with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero – the Irish Software Research Centre (www.lero.ie).

References

References

[1] S. Sahay, B. Nicholson, S. Krishna, Global IT Outsourcing: Software Development Across Borders, Cambridge University Press, 2003.

[2] J. Noll, S. Beecham, I. Richardson, C. NicCanna, A global teaming model for global software development governance: A case study, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE), IEEE, 2016, pp. 179–188.

[3] D. Šmite, C. Wohlin, Z. Galvina, R. Prikladnicki, An empirically based terminology and taxonomy for global software engineering, Empirical Software Engineering 19 (1) (2014) 105–153.

[4] P. J. Ågerfalk, B. Fitzgerald, H. H. Olsson, E. Ó. Conchúir, Benefits of global software development: the known and unknown, in: Proceedings of ICSP '08, Vol. 5007, Springer, 2008, pp. 1–9.

[5] J. D. Herbsleb, A. Mockus, T. A. Finholt, R. E. Grinter, An empirical study of global software development: distance and speed, in: Proceedings of ICSE 2001, 2001, pp. 81–90.

[6] S. Deshpande, I. Richardson, V. Casey, S. Beecham, Culture in global software development-a weakness or strength?, in: Proc of the 5th IEEE

International Conference on Global Software Engineering, IEEE, 2010, pp. 67–76.

[7] P. Ågerfalk, B. Fitzgerald, H. Holmstrom, B. Lings, B. Lundell, E. Ó. Conchuir, A framework for considering opportunities and threats in distributed software development, in: Proceedings of the International Workshop on Distributed Software Development, Austrian Computer Society, 2005, pp. 47–61.

[8] J. Noll, S. Beecham, I. Richardson, Global software development and collaboration: barriers and solutions, ACM Inroads 1 (3) (2011) 66–78.

[9] A. Avritzer, D. Paulish, Y. Cai, K. Sethi, Coordination implications of software architecture in a global software development project, J. Syst. Software 83 (10) (2010) 1881–1895.

[10] P. Ovaska, M. Rossi, P. Marttiin, Architecture as a coordination tool in multi-site software development, Software Process: Improvement and Practice 8 (4) (2003) 233–247.

[11] J. D. Herbsleb, Global software engineering: the future of socio-technical coordination, in: Proceedings of the Future of Software Engineering (FOSE '07), 2007, pp. 188–198.

[12] M. Conway, How do committees invent?, Datamation 14 (4) (1968) 28–31.

[13] A. M. D. Santana, F. Q. B. da Silva, R. C. G. de Miranda, A. A. Mascaro, T. B. Gouveia, C. v. F: Monteiro, A. L. M. Santos, Relationships between communication structure and software architecture: An empirical investigation of the conway's law at the federal university of pernambuco, in: Proceedings of the 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER), IEEE, 2013, pp. 34–42.

[14] M. Bano, D. Zowghi, N. Sarkissian, Empirical study of communication structures and barriers in geographically distributed teams., IET Software 10 (5) (2016) 147–153.

[15] S. Imtiaz, N. Ikram, Dynamics of task allocation in global software development, J. Softw. Evol. and Proc. (2017) 29. doi:10.1002/smr.1832.

[16] M. Babar, C. Lescher, Global software engineering: Identifying challenges is important and providing solutions is even better, Information and Software Technology 56 (1) (2014) 1–5.

[17] N. Ali, S. Beecham, I. Mistrik, 'Architectural knowledge management in global software development: A review, in: Proceedings of 5th IEEE Conference on Global Software Engineering (ICGSE), IEEE, 2010, pp. 347–352.

[18] S. S. M. Fauzi, P. L. Bannerman, M. Staples, Software configuration management in global software development: A systematic map, in: Proceedings of the 2010 Asia Pacific Software Engineering Conference, 2010, pp. 404–413.

[19] O. Sievi-Korte, S. Beecham, I. Richardson, Challenges and recommended practices for software architecting in global software development, Information and Software Technology 106 (2019) 234 – 253. doi:https://doi.org/10.1016/j.infsof.2018.10.008.

[20] A. Mishra, D. Mishra, Software architecture in distributed software development: A review, in: Proceedings of OTM 2013: On the Move to Meaningful Internet Systems: OTM 2013 Workshops, Vol. 8186, Springer Berlin Heidelberg, 2013, pp. 284–291.

[21] V. Clerc, P. Lago, H. van Vliet, Architectural knowledge management practices in agile global software development, in: Proceedings of the 4th IEEE International Conference on Global Software Engineering Workshop (ICGSEW'11), 2011, pp. 1–8.

[22] V. Clerc, Do architectural knowledge product measures make a difference in GSD?, in: Proceedings of the 4th IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 382–387.

[23] M. A. Babar, R. C. de Boer, T. Dingsøyr, R. Farenhorst, Architectural knowledge management strategies: Approaches in research and industry, in: Proceedings of Second ICSE Workshop on SHAring and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent 2007 (SHARK ADI 2007), 2007, p. 35.

[24] G. Borrego, A. Morá, R. Palacio, O.M.Rodriguez, Understanding architectural knowledge sharing in AGSD teams: an empirical study, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE), 2016, pp. 109–118.

[25] H. R. de Faria, G. Adler, Architecture-centric global software processes, in: Proceedings of the 1st IEEE International Conference on Global Software Engineering (ICGSE), 2006, pp. 241–242.

[26] F. Salger, Software architecture evaluation in global software development projects, in: Proceedings of OTM 2009: On the Move to Meaningful Internet Systems, Springer Berlin Heidelberg, 2009, pp. 391–400.

[27] M. Babar, A framework for groupware-supported software architecture evaluation process in global software development, J. Softw. Evol. and Proc. 24 (2012) 207–229.

[28] M. Babar, A framework for supporting the software architecture evaluation process in global software development., in: Proceedings of the 4th IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 93–102.

[29] M. Che, D.E.Perry, Evaluating architectural design decision paradigms in global software development, International Journal on Software Engineering and Knowledge management 25 (2015) 1677–1692.

[30] M. Bass, V. Mikulovic, L. Bass, H. James, C. Marcelo, Architectural misalignment: An experience report, in: Proceedings of The Working

IEEE/IFIP Conference on Software Architecture WICSA'07, 2007, pp. 17–17.

[31] R. B. Svensson, A. Aurum, B. Paech, T. Grschek, D. Sharma, Software architecture as a means of communication in a globally distributed software development context, in: Proceedings of the International Conference on Product Focused Software Process Improvement (PROFES 2012), Springer Berlin Heidelberg, 2012, pp. 175–189.

[32] S. Betz, D. ŠMite, S. Fricker, A. Moss, W. Afzal, M. Svahnberg, C. Wohlin, J. Borstler, T. Gorschek, An evolutionary perspective on socio-technical congruence: The rubber band effect, in: Proceedings of 3rd International Workshop on Replication in Empirical Software Engineering Research (RESER), 2013, pp. 15–24.

[33] J. Bosch, P. Bosch-Sijtsema, Coordination Between Global Agile Teams: From Process to Architecture, Springer Berlin Heidelberg, 2010, pp. 217–233.

[34] J. Herbsleb, R. E. Grinter, Architectures, coordination, and distance: Conway's law and beyond, IEEE Software 16 (5) (1999) 63–70.

[35] D. L. Parnas, P. C. Clements, D. M. Weiss, The modular structure of complex systems, IEEE Trans. Software Eng 11 (3) (1985) 259–266.

[36] J. M. Verner, O. P. Brereton, B. A. Kitchenham, M. Turner, M. Niazi, Systematic literature reviews in global software development: A tertiary study, in: Proceedings of EASE'12, 2012, pp. 2–11.

[37] C. Ebert, M. Kuhrmann, R. Prikladnicki, Global software engineering: Evolution and trends, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE), 2016, pp. 144–153. doi:10.1109/ICGSE.2016.19.

[38] O. Sievi-Korte, I. Richardson, S. Beecham, Protocol for an empirical study on software architecture design in global software development, lero techni-

cal report no. tr_2019_01, https://www.lero.ie/sites/default/files/TR_2019_01_Protocol_for_GSD_Arch_Design_Framework.pdf (2019).

[39] D. Cruzes, T. Dyba, Research synthesis in software engineering: a tertiary study, Information and Software Technology 53 (2011) 440–455.

[40] V. Braun, V. Clarke, Using thematic analysis in psychology, Qualitative Research in Psychology 3 (2) (2006) 77–101.

[41] M. Dixon-Woods, S. Agarwal, B. Young, A. Sutton, Synthesising qualitative and quantitative evidence: a review of possible methods, Journal of Health Services Reseach&Policy 10 (2005) 45–53.

[42] M. Burks, Y. Chapman, K. Francis, Memoing in qualitative research: Probing data and processes" (2008), Journal of Research in Nursing.

[43] K. Charmaz, Constructing Grounded Theory – A Practical Guide through Qualitative Analysis, SAGE Publications, 2006.

[44] M. Marinho, J. Noll, I. Richardson, S. Beecham, Plan-driven approaches are alive and kicking in agile global software development, in: Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2019, p. "to appear".

[45] M. J. Monasor, A. Vizcaíno, M. Piattini, Cultural and linguistic problems in GSD: a simulator to train engineers in these issues, Journal of Software: Evolution and Process 24 (6) (2012) 707–717.

[46] T. A. B. Pereira, V. S. dos Santos, B. L. Ribeiro, G. Elias, A recommendation framework for allocating global software teams in software product line projects, in: Proc of the 2nd International Workshop on Recommendation Systems for Software Engineering, ACM, 2010, pp. 36–40.

[47] V. Clerc, P. Lago, H. van Vliet, Global software development: Are architectural rules the answer?, in: Proceedings of 2nd IEEE International Conference on Global Software Engineering (ICGSE), 2007, pp. 225–234.

[48] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[49] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer-Verlag, Berlin-Heidelberg, Germany, 2012.

Dr. Outi Sievi-Korte is an Assistant Professor (tenure track) in Software Engineering at Tampere University, Finland. She joined then Tampere University of Technology at the start of 2009 to conduct her PhD thesis research on using meta-heuristics for optimized and automated software architecture design. She has since defended her dissertation in 2011, and received a Post-Doctoral Researcher grant from the Academy of Finland in 2014 for research in the field of global software development. Her research interests lie with data-driven software development, utilizing AI and meta-heuristics for software engineering problems, software design, software project management, and social aspects of software development. She has published nearly 30 peer-reviewed papers in software engineering. She is member of the board at Finnish Society for Computer Science since 2018, has attended program committees for many conferences, including the International Conference on Software Engineering: Software Engineering Education and Training track (2019), and co-chaired the Symposium for Programming Languages and Tools (2015).

Prof Ita Richardson is a Co-Principal Investigator in Lero and an Associate Professor in the Department of Computer Science and Information Systems at the University of Limerick. She leads the Process Quality Research Group, supervising projects on Global Software Engineering and Connected Health. She has over 200 publications, has supervised 15 PhD students to completion and is currently supervising 6 PhD students. Prof Richardson is an SFI Industry Research Fellow (2015-2017). She has received funding for her research from a variety of agencies including Science Foundation Ireland, Irish Research Council, European Union and Enterprise Ireland. Prof Richardson is UL's Athena SWAN champion, working at a national level for gender equality in science, technology, engineering and mathematics. She collaborates with a number of industries including IBM and Ocuco, and public bodies such as the Health Service Executive. She is on the editorial board of the Journal of Software: Evolution and Process, and serves on many conference programme committees including the International Conference on Software Engineering 2018 and the International Conference on Global Software Engineering over many years. Prof Richardson is guest editor of Journal of Software: Evolution and Process on Connected Health (2017), Journal of Software: Evolution and Process on Software Processes (2015), Expert Systems Journal on Knowledge Engineering in Global Software Development (2014), Journal of Software: Evolution and Process on Global Software Engineering (2012) and IEEE Software on Software Process for Small Enterprises (2007).

Dr Sarah Beecham is a Senior Research Fellow in Lero (https://www.lero.ie) the Irish Software Research Centre at the University of Limerick. Sarah joined Lero in 2009, to conduct research into software quality and process improvement. Her wider interests are in socio-technical aspects of software

engineering to include Software Engineer motivation, agile methods, distributed software development, and how technology influences the lives of the older adult. She works closely with industry, where her research is problem driven. She is also interested in the education of the next generation of software engineers. She has published over 60 peer reviewed papers in software engineering.

In her empirical research, Sarah employs both qualitative and quantitative methods. She has supervised and examined several national and international PhD students and has three students currently in the pipeline. Sarah sits on many Program Committees, has been general Chair for the Evidence and Assessment in Software Engineering conference (EASE), and was Co-chair for The Education and Training Track and ICSE (the premier International Conference in Software Engineering) in 2019, and is an associate editor for the Journal of Systems and Software.