Software Architecture Design in Global Software Development

Outi Sievi-Korte\textsuperscript{a}, Ita Richardson\textsuperscript{b}, Sarah Beecham\textsuperscript{b,*}

\textsuperscript{a}Tampere University of Technology, Finland
\textsuperscript{b}Lero - The Irish Software Research Centre, University of Limerick, Ireland

Abstract

In Global Software Development (GSD), the additional complexity caused by global distance calls for processes to help avoid project delays, ease communication and collaboration overhead, and improve control. How development tasks are broken down, shared and prioritized is key to project success. While many project management solutions are proposed, few guidelines exist that focus on how to architect and design software in distributed settings. In this study we address the question of how to design systems developed by distributed teams. We build on and augment findings in the GSD design literature by conducting an empirical study. Taking a qualitative approach, interviews with 13 GSD architects revealed several new challenges and best practices. Inter-rater reliability testing added a level of objectivity to our synthesis. All seven organizations were using Scrum in a distributed setting, which was found helpful to tackle some of the challenges found in the literature. Practitioners felt that designing software for distributed teams requires a new set of practices, particularly regarding adherence to defined processes. While many practices mirror those reported in the GSD literature, the inductive approach revealed new challenges, recommendations and even contradictions. The resulting set of guidelines and a conceptual model go some way towards supporting architects involved in GSD, who now have a more complete and cohesive set of warnings, which are, for the

\*Corresponding author

Email address: sarah.beecham@lero.ie (Sarah Beecham)
most part, matched by recommendations. We also have a better understanding of architectural practices in companies using Scrum development methods.

**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 particularly for larger companies [1]. Tasks are outsourced and/or off-shored [2] for a variety of reasons, mainly in order to reduce costs and gain access to local markets and resources [3]. 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 ‘distance’.

Distance [4] 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. Geographical distance makes it impractical to have face-to-face meetings, making teleconferencing and other substitutions necessary. Temporal distance, if not handled correctly, often results in delays, as there might be very little overlap in office hours between sites, making it even more difficult to agree on meeting times. Even asynchronous communication, such as emails, suffer from temporal distance, as a simple question may take until the next working day to be answered. Finally, socio-cultural distance brings a certain flavor to the meetings between different sites, and good communication skills are required to fluently execute meetings and allow projects to advance.

Global distance thus calls for more effort in terms of inter and intra team communication, coordination and control [5]. Working communication methods need to be in place to overcome the challenges brought about by distance. Projects need to be especially well-coordinated [6], 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 [7, 8]. Finally, especially due to the
differences in working cultures, more rigorous control over activities is required.

Due to the aforementioned aspects, GSD is a particularly challenging form
of practicing software development. A common way to alleviate the challenges
is to minimize required communication between sites. Reducing the number
of different sites that need information from each other to perform their tasks,
leads to fewer meetings, fewer emails sent and fewer misunderstandings due to
cultural differences. Further, it is commonly acknowledged that to achieve this
optimal communication level, the key is to allocate tasks so that there are as
few dependencies between sites as possible [9].

Tasks are directly derived from the software, and dependencies between sites
are thus a direct result of dependencies within software. Dependencies within
software, in turn, are dictated by the software architecture - how the software
has been designed. Conway’s law [10] states that the organization’s structure
and the software architecture will end up mirroring each other, and this has
been validated by many studies over the years [11, 12, 13]. Thus, it would seem
that by simply creating a modular architecture that follows the organization’s
structure and available skills would solve a lot of issues with GSD [6].

Software architecture design, however, is a very complicated activity. In ad-
dition to “simply” considering the modular structure of the software, architects
need to consider required technologies and dependencies between them, available
resources, available budget and schedule, customer requirements and pressure
from the marketing department - just to name a few points. The modularity
of a software itself is not straightforward either, if there are concerns spanning
multiple layers and components. If tasks are divided by layers, is it clear where
the boundaries between layers lie? 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 ex-
ist 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 [14], 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. [15, 16]. However, many of these studies present secondary results from synthesising or mapping architectural reviews and architectural knowledge management issues in GSD. Very few directly investigate how to perform software architecture design in a distributed setting.

In this paper we present an empirical study of software architectural approaches to developing software in GSD projects. Thirteen practitioners from seven organizations were interviewed. This study builds on previous work, particularly on our in-depth systematic literature review (SLR) on architectural challenges and practices in GSD [17]. The aim of this study is to validate findings from our SLR and provide new insights to our research question, What are the challenges faced and practices used by practitioners involved in software architecture design in GSD? In our SLR we found challenges particularly related to change management, quality management and task allocation, for which there were no practices found in the literature, and were hoping to complement the set of recommendations particularly related to these aspects.

This paper is organized as follows: Section 2 presents the background to this study through a summary of the related work on software architecting in GSD. In Section 3 we outline our empirical research method and in Section 4 we summarize the results from the practitioner interviews. Section 5 presents practices and guidelines for software architecting in GSD. In Section 6 we compare and discuss our results with those given in the related literature. Finally, in Section 7, we consider threats to validity and summarize our contribution.
2. Background

2.1. Related Work

Software architecture related studies in a GSD context were reviewed by Mishra and Mishra [18] who viewed architecting in terms of either knowledge management (see, e.g., [19, 20, 21, 22]) or process and quality (see, e.g., [23, 24]).

Additionally, there are several studies on performing software architecture reviews and evaluation 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 [25, 26]. Evaluation of software architecture decisions, in turn, has been studied by Che and Perry [27].

Further, Conway’s law features widely in the related literature, where architectural issues have been addressed in relation to task allocation and coordination of GSD projects (see, e.g., [28, 29, 30, 31]). Herbsleb and Grinter [32], 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 in the 1970s by Parnas [33].

There have been several systematic literature reviews in the area of GSD in general, as revealed by the tertiary study by Verner et al. [34]. Based on this study, it can clearly be seen that organizational factors, engineering or the development process, and software project management issues are the most studied areas in GSD. Notably, from the listed 24 SLR studies, only one involving design is listed. This is a review concentrating on architectural knowledge management (AKM) issues by Ali et al. [15], 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. [15], several studies consider software construction [16], but they take a process viewpoint. This strongly suggests that there is a gap in design related research within GSD. This mismatch between industry needs and research conducted was identified in an evaluation of 10 years of research and industry collaboration in Global Software Engineering [35]. 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 Overview

In 2018 we conducted a systematic literature review (SLR) on software architecting challenges and practices in GSD [17]. The SLR synthesis enabled us to construct a concern framework of challenges and practices and present a conceptual model of architecting in GSD. This conceptual model is given in Figure 3 where the challenges and practices are grouped under themes. Relationships between the themes are also shown. Themes (concepts) are presented as classes, practices and challenges are given (in condensed form due to space restrictions) as class members (coded with SLR-P1 – SLR-P9 for practices and SLR-C1 – SLR-C9 for challenges), and different themes have relationships between them. We have used the directed labeled association to mark the cases where the concepts have indisputable relationship between them. We have used the directed dependency notation where the relationship between concepts is clear but how much actions regarding one theme affect another depends on the case organization and project. Finally, inheritance is used to notate a spe-
cial relationship between themes and directly derived sub-themes. Additionally, two core concepts of architecting (*Design Decisions* and *Project management*) are noted with stereotypes to distinguish under which core concept each theme falls. Classes where these concepts overlap are marked with a special stereotype "Design decisions and Project Management”.

As shown, the practices and challenges are related to the following key themes: Organization (Structure and Resources), Ways of Working (AKM, Change Management and Quality Management), Design Practices, Modularity and Task Allocation. This model, in addition to the checklists we also developed as a result of conducting the SLR, gave us a starting point for our interviews. We focused on the key SLR themes when constructing our questions ensuring that the topics were found to be challenging in the literature were covered, and where possible we probed the interviewees to elicit practical examples of the practices listed in the literature.

### 3. Research Method

We have two research questions to solve in our study.

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

**RQ2**: What kind of practices are used by software architects to accommodate the distributed nature of the development work?

#### 3.1. Method and interviewees

To answer our research questions, we performed semi-structured interviews with 13 representatives from seven different companies with headquarters in three different countries. All representatives participated voluntarily. All interviews were performed by the first author, who recorded the interviews and wrote notes. The interviews lasted between 1 hour and 2.5 hours. The purposive sample of interviewees all had experience of working in distributed software development projects working with architectural issues. Some had additional
experience from project leadership or management. Interviewee and company backgrounds are summarised in Table I. Companies are coded with letters A-G. As shown, in each of 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 interviewed practitioners worked in very similar projects or roles, while in company G the practitioners had much more varying roles, though all related to architecting. Note, that “Number of sites” indicates the number of different sites that are or have been involved in the projects where the intervie-
wee in question has worked on (i.e., sites per project). Different countries, in turn, indicates in how many and what countries the different sites have resided in considering all projects.

3.2. Questions

The interview questions were based on the themes found in our SLR [17]. We ran a pilot interview with a colleague (not involved in the study) having several years of experience in software architecture research and teaching. Based on the pilot interview some questions were clarified and points where possible clarifications might be needed during the interview were noted. The interview proceeded in four phases.

**Phase 1:** Explaining to the interviewees the purpose of the study and the background of the field, defining, for example, the terms "GSD" and "Software architecture design" to ensure a common context.

**Phase 2:** Background information. We asked interviewees about their and the company’s background, as given in Table 1. Further, we asked about common issues often related to GSD, such as, *do the teams have a common language? what are the reasons behind distributing development work?* and, *what are the biggest GSD challenges the interviewees have found?*

**Phase 3:** Architecting in GSD. In this phase we attempted to unearth architecting practices by asking open questions about the principles, practices and guidelines that the practitioners had followed or found useful (or not) in their work. Our motivation for these open questions were to discover new challenges and practices that have not been previously reported in the literature. We further asked about how much effort was usually put into architecture design and what kind of backgrounds the architects had, as these were issues discovered in the SLR.

**Phase 4:** Themes. Based on our SLR, we identified gaps in the literature related to architecting in GSD. As shown in our conceptual model in Figure 3, there are a variety of challenges for which there were no answers found in the literature. Further, where the practices do address the challenges, they often
Table 1: Interviewees' backgrounds

<table>
<thead>
<tr>
<th>Field of company</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of participants</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Field of company</td>
<td>Power/Electrical automation</td>
<td>Software Development</td>
<td>IT Consulting / Software Development</td>
<td>Software development</td>
<td>Mining/ machinery and information systems</td>
<td>Telecommunications</td>
<td></td>
</tr>
<tr>
<td>Size of company (employees)</td>
<td>500 000</td>
<td>30+</td>
<td>100</td>
<td>50-60</td>
<td>1000</td>
<td>35 000</td>
<td>100 000</td>
</tr>
<tr>
<td>Size of IT/Software Development section</td>
<td>Most activities involve sw</td>
<td>20+</td>
<td>30</td>
<td>90%</td>
<td>70%</td>
<td>100</td>
<td>30%</td>
</tr>
<tr>
<td>Number of sites</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3-5</td>
<td>2-5</td>
<td>3-4, 2-5, 12</td>
</tr>
<tr>
<td>Number of countries</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3-5</td>
<td>5</td>
<td>2-3, 2-5, 9</td>
</tr>
<tr>
<td>Countries</td>
<td>USA, IT, IN, FI, PK</td>
<td>NL, HR, FR, SA</td>
<td>FI, VN, DE, JP</td>
<td>IE, FI, IN, PL, RO, AR</td>
<td>IN, FI, NL, USA</td>
<td>FI, CN, IN, PH, PL, DE, USA, FR</td>
<td></td>
</tr>
</tbody>
</table>
mirror the challenge. Thus, we had identified a need to focus on the gaps and issues discovered in the SLR to find out how challenges are addressed in practice. Also, through these open questions we were in a good position to uncover new challenges not yet found in the literature. We hence utilized our concern framework to create detailed questions around the themes where we found significant challenges in the literature. The themes are as follows:

1. Practices and design agreements (including well-defined interfaces)
2. Aligning software architecture, work structure and resources
3. Communication and lack of awareness between sites
4. Identifying dependencies
5. Creation of and sharing architectural artefacts
6. Quality assurance
7. Design compliance

3.3. Analysis

The interview recordings were transcribed (and translated into English, where necessary), by an independent transcribing and translator company. Due to the interview questions’ design, conventional content analysis seemed unsuitable as an analysis technique. As we had very specific themes in Phase 4 of the interview, we would unsurprisingly have found recurring words and phrases repeating the wording of the themes. Because of this, we did not count word frequencies, and instead opted for a form of thematic analysis [36, 37, 38] accompanied by memoing [39, 40]. The analysis and validation process is depicted in Figure 2.

When coding the transcripts, the codes “practice” and “challenge” were predetermined, but other codes were generated based on the material resulting in 35 different codes. After creating codes, we went through the material again creating memos, based on which 18 higher level themes were found. A combination of the coding and memo themes was then used to extract concerns, formulated as practices and challenges.
We produced a three-level challenge-practice framework containing 26 individual concerns related to practices and 23 individual concerns related to challenges. The framework was validated so that a set of 10 quotes from the interviews (for both challenges and practices separately) were given to two researchers, and these quotes were mapped to the challenges and practices at all three levels. The validation process resulted in two major revisions to the framework. After each revision the framework was revalidated. In the final version there are 22 concerns related to challenges and 17 concerns related to practices. We will present the framework in detail in Section 5. The concerns are mapped to higher level themes together with practices and challenges found in our SLR to provide a combined framework and model on software architecting in GSD. We arranged all the concerns in the SLR together with the concerns found in this study under a new set of practices and challenges, finding eight of each. Practices were then mapped to challenges and the conceptual model was revised, portraying combined knowledge from the literature and our empirical study. All stages were done so that two researchers individually performed the analysis, revised until agreement was reached, which was further validated by a third researcher.

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 wanted to know what kind of software processes
were followed, what were the drivers behind distributing development, and what kind of issues the practitioners had found with the distributed setting in general. We have collected answers to such background questions in Table 2. We have condensed some repeating answers for the last question due to space as follows:

- **COMM**: How to handle communication
- **CULT**: How to handle cultural differences
- **TIME**: How to handle different time zones

Firstly, 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) and about the same (5 hours) as other sites. 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 as significant.

Secondly, 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 bought a smaller local company to gain access to a required resource.

Thirdly, we note that all the companies are using or at least 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.

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

4.2.1. Role of Architect

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. Firstly, 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 also 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), as the whole company is based on an ideology that there is no need for people to work in the same premises in order to get the job done. Companies B and F, in turn, have the least number of sites (only 2 active sites currently) and the lowest number of different teams involved in development.

Secondly, 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 (as in G, where there are different level architects for feature, component and product line), the experience of an architect’s role and responsibilities is more subjective.

Finally, there seem to be two main practices on how to fit architecture design into the (varying) agile practices. One option is to really do agile architecting so that architecture design evolves 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 “sprint zero”, where the main portion of the architecture is designed before development actually starts. This is often required by the customer.

4.2.2. Issues and Recommendations

We investigated the issues which occur when doing architectural design in practice. We queried how issues are solved by asking open questions on what the interviewees considered to be the biggest issues and the most important things to consider when making design decisions in distributed development.

Issues

In Table 2 we listed the most important challenges which participants faced in GSD in general. However, we need to dig deeper to discover the most important issues affecting architecting work and the ability to design high-quality software in a distributed environment.

The interviewees raised above all the following issues:

- Keeping architecture compliant with organization structure
- Handling instability
- Difficulties due to distances
- Understanding architectural decisions
- Achieving modularity and separation of concerns
- Lacking knowledge management practices
- Deviating from processes

Compliance with organization structure is directly what Conway’s law warns about. We illuminate this challenge with a quote from a practitioner: "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
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of participants</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Who is in charge of decisions</td>
<td>Architect team. Members from all sites.</td>
<td>CTO, who leads a decentralized team of architects, each in charge of one sub-area.</td>
<td>Two central architects in charge of whole product, one architect representative in each developer team.</td>
<td>One Chief architect for each project</td>
<td>Multi-site architect teams, and individual architects from the teams. One team per sub-area.</td>
<td>One chief architect</td>
</tr>
<tr>
<td>What are the responsibilities of an architect</td>
<td>N/A</td>
<td>Responsible for the design of their own sub-areas, finding out what should be done, making a preliminary plan on how should be done.</td>
<td>Architectural decisions, weighing the balance of trade-offs, different stakeholder requirements. Safeguarding the implementation, communication.</td>
<td>Chief architect is responsible for the big picture</td>
<td>Linkage between company goals and how software is developed. / Doing research on options. Estimating risks. Reversibility</td>
<td>Chop the product into the right kind of divisions, clear and reusable parts.</td>
</tr>
<tr>
<td>How is architecture design fitted to Agile processes</td>
<td>N/A</td>
<td>Architecture designed before project starts.</td>
<td>Treated as normal development work, architectural tasks are tickets for the PO to prioritize and distribute.</td>
<td>Project started with a month long design period, high-level architecture and prototype done.</td>
<td>Architecture deliverables for each sprint, some design before development. Product is agile, agile architecting under discussion.</td>
<td>Plan is to have architecture planned 2-3 sprints ahead of time. Doesn’t always happen.</td>
</tr>
</tbody>
</table>
same product, but they belong to the different, they are reporting to the different editors”.

Adding to the challenge of keeping the architecture compliant with organization is the proneness to instability, which is particularly emphasized 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.

Communication is further 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 mentioned by one of 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.”.

Difficulties in communication also result in lower commitment. When the developers at remote sites are not given the kind of responsibility they would want, the tasks do not match their skills, or they are not included in the meeting where decisions are explained, they are less committed to the assigned work.

The aforementioned problems in communication and practical work easily leads 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 software being developed. What is most alarming is that these conflicts may not be found before they have already caused errors in development. A practitioner discusses the issue on 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.”. The effects of disagreements is 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 here 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.”. This kind of experience shows how arguing over the architecture design, or worse, deliberately implementing the software so that it does not concur with the design, can bring about serious problems and further emphasize the difficulties on separating concern and achieving modularity.

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

Finally, practitioners 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. "if there is a situation that something has happened over there, that they’ve done something that’s a bit different from the original plan, the end result isn’t quite what we’d hoped for originally, and then we make a request that they’d start to change it back, then it’s... Then they’ve just moved the task forwards and forwards and forwards despite the request”.
Recommendations

Through this research, we have identified concrete recommendations regarding considerations in architectural design. We have divided them into two categories: Architecting practices and Task allocation practices. Architecting practices are:

- Determine driving architecture style
- Determine platform to base design on
- Focus on requirements
- Consider existing product and its constraints
- Create a proof of concept
- Create demonstrations
- Create microservices to separate development items
- Create product boundaries based on APIs
- Apply continuous integration

These recommendations have stemmed from three different drivers. First, there needs to be a commonly known baseline that forms the backbone of the architecture. This would include determining the architecture style or platform, considering requirements and constraints and creating quick proofs of concept and demos to ensure common understanding. Secondly, there is a need to separate development items so that dependencies are de-coupled, thus alleviating the problems of distributed development – solutions to these are found in microservices and API boundaries. Finally, continuous integration is suggested as a tested solution to ensure the synchronization of development of distributed components.

The architects then also had advice which fell more into the realm of task division once the architecture already exists:
• Base task division on layers
• Consider maintenance responsibilities as a driver for task division
• Keep development of core product at one site
• Avoid leakage of site-specific functionalities between sites
• Have clearly separated responsibilities to different sites

The common ideology, unsurprisingly, is that clear boundaries should identify how tasks are divided. The best place to draw the line between two sites depends on the product and the architecture - in some cases it is layers, in some cases it is who will be doing maintenance, and in some cases it is the structure of the product. The most notable fact is, however, that while we asked for most important factors in design decisions, over a third of the answers actually considered the situation where the architecture exists and tasks need to be divided based on it.

Additionally, we were able to elicit some recommendations on managing the architectural design process in general:

• Establish practices to enhance knowledge distribution across sites
• Have clear roles to aid in governance
• Align architecture and organization

To truly facilitate distributed development, one should not remain content with simply having mechanisms in place that enable knowledge distribution, but rather create a working culture where the need for increased communication between sites is recognized and even enforced. One mechanism to accomplish this is to engage developers from all sites into the design work, as discussed along with our findings on the role of architect.

Practitioners found that engaging developers increased their understanding of architectural decisions and commitment A practitioner discussed how they have utilized developer engagement: "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.”. Practitioners 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 mismatches in assumptions.

In addition to engaging developers, managing the design can be aided by assigning specific individuals to distinct tasks and having clearly defined responsibilities. In each of the companies where we interviewed, they had either a team or a person responsible for architectural work. This is a significant discovery, as all companies also applied Scrum, where the assumption is that teams are self-organizing and thus even architectural work could be done independently within teams. However, in our study, practitioners strongly supported having someone with the final say on architectural decisions. Further detailing the architect’s role, it was advised that architects handle all relevant communication between different stakeholders. Additionally, there should be a clearly named person in charge of managing knowledge distribution, architectural work and prioritization. Often it was the case that people assumed that someone (a team leader or just some active individuals) would take care of knowledge management, and then it was left unclear whether actually all relevant individuals had been informed.

Finally, our material further confirms Conway’s law, as practitioners highlight the need to align the organization with the software architecture. The architecture may act as a driver, and resources are acquired based on what kind of skills and how much workforce is needed to implement the designed architecture under the given schedule. Vice versa, the organization may act as a driver, and the architecture design is based on the skills and other resources currently available in the organization. These two approaches together illustrate the mixed responses we received in our study. One practitioner when asked, whether the architecture drives the organization or the organization drives the architecture, stated: ”It’s an evolution”.

22
5. Framework for Software Architecture Design

As discussed in Section 3, we elicited detailed concerns from the interview material. Concerns were linked to either challenges faced by practitioners or practices they recommended. To create a framework for software architecting in the context of global software development, we have combined our findings with the concerns found previously in our SLR, to provide the most complete set of recommendations available. We further mapped the practices to challenges and refined the conceptual model to represent architecting in GSD. In the following we will first present the updated model of practical architecting and mapping of practices to challenges. We will then discuss the found practices and challenges in more detail and list the associated concerns. The set of concrete concerns may be used by practitioners as a way to spot possible challenges and prepare them to solve challenges already present in projects.

5.1. Conceptual Model

We mapped the presented challenges and practices to a conceptual model, using the SLR-based model as a baseline. The conceptual model for architecting in GSD is given in Figure 3 using the same notation as for the SLR-based model (presented in Sect. 2.2). 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. As combining the results from the empirical study with those found in the SLR lead to a more complex set of practices and challenges, the conceptual model has also been modified. Further, how practices map to challenges is more complex. Practices that are under the same theme as a corresponding challenge in the model are natural solutions to that challenge. However, practices that are associated with challenges via relationships in the model can also be helpful. The interpretation of relationships is further illustrated in Figure 4. For the sake of clarity the complete mapping of practices to challenges is also given in Table 4.
The model we published through our SLR has now evolved by including the empirical study described in this paper, and we identify particular differences. Firstly, we added new relationships between Project Management and Role (of Architect), and Role and Architect, as these concepts were highlighted during the study. Secondly, we modified the model so that Task Allocation was placed...
Figure 4: Illustration of relationships

as a sub-theme under Ways of Working instead of appearing on its own as before. Moreover, the relationships between Task Allocation and other concepts were modified to better reflect current practices. Thirdly, the relationship between Design Practices and Design Decisions has been modified so that now Design Decisions are part of Design Practices. Finally, the relationship between Project Management and Ways of Working was fortified to be a clear association instead of depending on the Organization. Further, the relationship between Ways of Working and Design Practices now works both ways.

The most significant updates to the SLR-derived model are thus the identification of increased dependencies on the architect’s role and how task allocation fits into the model. In our empirical study we found that the role of architect is
dictated by project management practices. Organizing architecting work to one chief architect, architects on several levels or distributing the work to a team of architects, perhaps involved with development as well, has a large impact to what architecting means in each particular case. Thus, the role determines the tasks in which an architect is involved. 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 a 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

The 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.
Table 5: Design Process and Quality Management

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td><strong>Challenge:</strong> Mismatch between organizational structure and architectural design and difficulties in dealing with instability</td>
<td>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)</td>
</tr>
<tr>
<td>C3</td>
<td><strong>Challenge:</strong> Lack of control over software quality</td>
<td>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)</td>
</tr>
<tr>
<td>P6</td>
<td><strong>Practice:</strong> Align architecture with organization arrangement</td>
<td>Include business goals in design (P6_co1_slr) Base architectural decisions on available resources (P6_co2_emp) Establish quality management practices (P6_co3_emp)</td>
</tr>
</tbody>
</table>

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. An additional aspect to this challenge, however, is that organizations are seldom fixed, but their structure is unstable. The concerns brought forward by the practitioners (C1_co2_emp, C1_co3_emp) are very similar to those already found in the literature - matching the architecture and organization is found difficult.
The Quality Management related challenge (C3) highlights the need for proper quality assurance. While we previously found some concerns related to deteriorating quality (C3_co1_slr, C3_co3_slr) and general problems in quality management (C3_co2_slr) in our SLR, new concerns were brought to light by practitioners. First, architectural reviews, where quality is often considered, are more difficult to arrange in a distributed setting (C3_co4_emp). However, in distributed projects such reviews would be even more beneficial than in a collocated one, as our practitioners reported concerns on insufficient recording of quality requirements (C3_co6_emp). Finally, different sites may have different aptitudes for running automated tests, which is considered one of the key issues in quality assurance by our practitioners (C3_co5_emp). These concerns are also addressed as part of P6 – which raises quality management practices as a separate concern (P6_co3_emp) when aligning architecture and organization. The detailed practices found in our empirical study mention arranging architectural reviews and having good testing coverage.

We recommend aligning architecture with organizational arrangement (P6) in addition to purely aligning it with the organizational structure. Organizational arrangement means the processes, practices and resources that are set up in the organization at large. Our practitioners particularly highlight the need to base decisions on available resources (P6_co2_emp) – here resources means the developers, the effort they can put into their work, the skills they have, what kind of technologies they have experience on, and where they are located. It also means access to hardware, software licenses, and other necessary resources. 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. Poor AKM (C2) is thus quite often experienced and demonstrates itself in many ways. Proper knowledge management entails ensuring that all sites have access to the documentation they need to properly carry out their tasks 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 all these systems, who actually accesses them, and if someone does access the documents, is the system actually 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 further 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 a variety of problems when developers unnecessarily attempt to duplicate their functionality (C2_co12_slr).

One concern from 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 with it raising issues about potential drawbacks of certain choices is a natural part of architecting, the concern here is specifically highlighted in the distributed setting due to difficulties in communication and
<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/ Practice</th>
<th>Concerns</th>
</tr>
</thead>
</table>
| 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) |
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 end up questioning the selected solutions.

To handle these challenges we recommend communication of architectural decisions to all stakeholders (P1) - a practice that should be self-evident to capable architects. However, the more detailed concerns may help notice gaps in how communication is handled. It is not enough to simply put information out there, but architects (or a team of architects 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 actually enhances communication (P1_co1_emp), such as visiting remote sites and having common fixed meetings.

Concerns related to software development governance (P3, see Table 10) may also aid in improving knowledge management, particularly those recommending 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 approachable than others.

5.2.3. Ways of Working

How and what kind of practices are established in design process and development are defined in Ways of Working. In our SLR we discovered several concerns related to insufficiently defined practices or how practices were followed across sites (C4). To address this is to have a standardized set of practices across sites (P5). This essentially means that all sites and persons involved in architecting work should have a common agreement on what particular practices and drivers are applied in design (P5_co1_slr), which is not a given in distributed
Table 7: Shared Practices

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
</table>
| 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) |

Projects. However, in our empirical study we were able to find further solutions to how related concerns could be handled. We discovered a concern related to ensuring fit to requirements (P5_co6_emp). Architecture design always begins from eliciting functional and non-functional requirements, and creating the architecture that answers to them. However, particularly when architecting is distributed, if the design work is not well-coordinated, the original requirements may easily fade into the background, resulting in compliance issues in the long run (C4_co5_slr). This may be aided by utilizing agile methods (P5_co4_emp) - handling a smaller set of requirements at a given time and being able to quickly adjust development work in an unstable organization will aid handling compliance and communication issues, and can often help discovering misunderstandings more quickly.

Ways of Working can be further improved by having 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, there are bound to be challenges. The most notable ones 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), practitioners note how the complexity of the architecture brings challenges to the design (C5_co2_emp). The more complex a product in question, the more challenging the design is, no matter how the project or organization are 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 SLR (C5_co1_slr, C5_co5_slr), in our empirical study such concerns were complemented by challenges faced by the practitioners: 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 recommendations on designing based on practices particularly suitable for GSD. In more detail, we found several practical concerns related to modularity and separation of concerns in the architecture (P2_co2_emp and P2_co4_emp). Our practitioners 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 practitioners also stressed the possibilities of reuse (P7_co3_emp), which is
<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>C5</td>
<td>Challenge: Problems associated with architectural design decision and identifying dependencies</td>
<td>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)</td>
</tr>
<tr>
<td>P2</td>
<td>Practice: Apply architectural design practices for global software development</td>
<td>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)</td>
</tr>
<tr>
<td>P7</td>
<td>Practice: Do continuous improvement</td>
<td>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)</td>
</tr>
<tr>
<td>P8</td>
<td>Practice: Use various architecting modeling techniques</td>
<td>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)</td>
</tr>
</tbody>
</table>
also easier if the design is modular. Considering 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 (P8), as a variety of models may aid in understanding the decisions. Additionally, design is aided by general architecting practices as given in P5 (see Table 7).

5.2.5. Task Allocation

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>C6</td>
<td>Challenge: Issues with task allocation in a distributed setting</td>
<td>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)</td>
</tr>
<tr>
<td>P4</td>
<td>Practice: Implement architecture-based task allocation in global software development</td>
<td>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)</td>
</tr>
</tbody>
</table>

Task allocation in a distributed setting (C6) easily becomes challenging for
a variety of reasons. Modular design is highly stressed in GSD, as task allocation is often based on the assumption that modules or concerns are clearly separated and decoupled. However, if dependencies between tasks and subsequently between teams are not identified, work easily becomes very challenging (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 usually kept separate between sites in a distributed setup, it often becomes the case that two sites are developing large modules which ultimately need to be fit together for the final product. If one module is delayed, development on the other will, at some point, also come to a halt, as certain testing will need integration to the offshored module (C6_co4_emp).

Based on our combined data synthesis we recommend an architecture-based task allocation (P4). This is already supported by the literature (P4_co1_slr, P4_co2_slr, P4_co5_slr). Practitioners 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). Another option of ensuring alignment between the organization and architecture is to allocate tasks so that resources at a given site actually match the task given to 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 showcase as not considering organization management in the design process (C7_co2_slr) or in dividing tasks (C7_co1_slr). There are often also problems in knowledge management when poor governance has resulted in bottlenecks (C7_co7_slr) or lack of expertise in design work (C7_co6_slr). Our practitioners also noted problems related to inequality
Table 10: Governance and Processes

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
</table>
| 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 \((P3\_co2\_slr)\)  
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 \((P3\_co5\_emp)\) |

Practitioners further reported problems related to how processes are followed. In some cases practitioners 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, practitioners 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 to address the aforementioned issues is to engage developers across sites in architectural work (P3_co5_emp). Engaging developers from various backgrounds and sites will aid in condensing of knowledge and finding expertise. Similar benefits regarding knowledge management can be achieved with having clearly appointed people for different roles (P3_co1_emp). Further, practices found in the literature (P3_co1_slr, P3_co2_slr) related to how tasks and changes spanning across sites should be handled may aid in concerns related to organization.

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 particularly related to concerning 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 need to note a gap in how the found practice and the related concerns address concerns raised particularly by the practitioners. We did not find particular concerns that would directly aid in issues related to processes.

5.2.7. People Management

<table>
<thead>
<tr>
<th>ID</th>
<th>Challenge/Practice</th>
<th>Concerns</th>
</tr>
</thead>
</table>
| 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) |
Managing people and associated soft issues presents a clear gap in our studies. This is unsurprising, as we were not particularly searching for recommendations on how to handle these kinds of issues. What is slightly surprising is that this challenge was so strongly brought forward in our interview material. While we were concentrating on unearthing the essentials to design software architecture in a distributed setting, and were focused on finding recommendations for constructing the architecture on the drawing board, the practitioners experienced the lack of commitment in various ways (C8_co1_emp, C8_co2_emp) as ‘very disturbing’ to ‘executing the design’. Further, many practitioners told of problems raised by cultural differences and particularly when reporting progress (C8_co4_emp) – in some cultures it was customary to try to maintain a good impression (by repeatedly answering that work is progressing even though it was not) for as long as possible, while in some cultures a more straightforward atmosphere was appreciated (openly discussing delays and difficulties). People easily became frustrated when they were expecting a certain status of completion and then at the last minute it became apparent that this was not achieved.

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. Particularly 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 practitioners gave this particular recommendation is strongly linked to commitment and motivation – giving a feeling of responsibility.

6. Discussion

6.1. Architecting in GSD

In this section we discuss and summarise how our study addresses research questions relating to the challenges practitioners face in software architecture design in GSD, and how they are solving those challenges.
Our empirical study revealed challenges not previously noted which related to the design process and issues stemming from the organization. Regarding design choices, the concerns were quite clearly emphasized by the distributed nature of software development. When tasks are distributed, finding and defining logical entities and interfaces between them, and identifying dependencies within the architecture are critical. However, when the design is also distributed, tackling such challenges is difficult.

The practices utilized are mostly based on a common denominator: agreement. Agreeing on management practices and collaboration, agreeing on common design principles, reaching an agreement on roles for different tasks and making sure that the organization and architecture are in agreement (i.e., aligned). Taking a closer look at the commonly agreed principles for architecture design, there is the expected emphasis on modularity and loose coupling. However, the practitioners also have a clear motivation to emphasize reuse of components and consider the long-term affect of their choices. 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. Paying attention to reuse and long-term effect should aid in unstable environments - with a library of small, reusable components there is a solid foundation to build on regardless of what sites are available, and considering long-term effects, such as dependencies on frameworks, libraries or hardware, will further ease design in the future.

The motivation for conducting the empirical study was to broaden our understanding of architectural design methods applied in distributed software development. While the SLR we conducted [17] illustrated general problem areas and lessons learned, we were uncertain as to the completeness or consistency of our results. Both our empirical study and several sources in the SLR (e.g., [41, 42, 43, 28]) identified misalignment between organization structure and the software architecture as one of the biggest challenges. In the SLR we only found practices that were mostly architecture-driven such as keeping tightly coupled items at the same site as suggested by, e.g., Mockus and Weiss [41] and Pereira
et al. [44], or by ensuring loose coupling between components dispersed to distributed teams, as suggested by Bosch and Bosch-Sijtsema [31] and Salger et al. [24]. However, our empirical study revealed that, in practice, there is more support to the opposite approach of using the organization as a baseline and aligning the architecture to resources. Most notably, the empirical study showed that aligning the architecture and organization evolves over time, where one or the other is often changing.

Both our empirical study and the SLR also revealed similar challenges regarding communication and knowledge management. The empirical study, however, highlighted how much these challenges reflect on the execution of daily tasks, and how much the challenges are due to differences in working culture, while the SLR focused more on the technical challenges of communication, such as how electronic systems were used [22, 45] or difficulties in effective creation and sharing of artefacts, as raised by, e.g., Sangwan and Ros [16], and Che and Perry [27]. Similarly, based on the SLR, we were only able to provide either very general solutions, such as ”communicate clearly” or utilizing a team of architects (as suggested, e.g., by Vaikar et al. [47] and Paasivaara and Lassenius [48]), or quite technical solutions, such as using a single repository for architectural artefacts, (see, e.g. Sauer [49] and Cataldo et al. [50]). However, in the empirical study, the focus was on establishing collaboration practices across sites – and notably, involving others in addition to architects – and having clearly defined roles for central people. Similarly, both studies raised as an issue how architectural decisions were made, and solutions were also similar - apply commonly applied practices and principles. Further, both studies raised the issues of keeping the architecture design modular. However, while it was acknowledged that modularity is a desired aspect, and in the SLR, e.g. Pereira et al. [44] and Clerc et al. [51], recommended the use of well-defined interfaces, our empirical study revealed that finding the correct boundaries for such interfaces is sometimes very challenging. Furthermore, the SLR design challenges were discussed more from the perspective of insufficiently defined practices (by, e.g., Ovaska et al. [8], Clerc [20], and Popovic et al. [45]), while the empirical study brought
out challenges related to complexity of the software and disagreement between design choices.

Salger et al. [24] and Clerc et al. [51] found challenges related to stability of architecture, and, e.g., Betz et al. [30] noted the lack of compliance between the architecture and requirements, processes and organization. While we specifically asked our interviewees about these challenges, we did not find concrete solutions - nor did they consider them a particular challenge in their work. It would seem that either the architecture was found stable enough or instability was considered the normal state, or that stability and compliance to business processes etc., is such an implicitly evident part of the design process that the interviewees could not really define how they approached the issues.

Both the empirical study and, e.g., Bass et al. [28] and Sangwan and Ros [46] found as a challenge the lack of methods for quality assurance. However, neither the SLR nor our empirical study could provide concrete practices for this challenge. Additionally, in the empirical study we found difficulties in following processes a major challenge in the industry. This did not come up in the SLR, but the practitioners raised the issues of how practical work was difficult to fit to processes, how processes suffered when people were not committed to them (usually at remote sites) and when they had to prioritize schedule over the defined processes.

6.2. Threats to Validity

We will consider threats to validity as described by Wohlin [52] 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, on 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.

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 and External Validity

Construct validity concerns how well the results are generalizable. 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 uncover practical architecting challenges and practices follow-up questions were almost always required.

External validity, in turn, concerns how well the results are generalizable to the industrial practice. As this study was conducted with a cross-section of practitioners currently working in the industry, we argue that the results are generalizable in the field of GSD.

7. Conclusions

In our previous work [17] we identified several challenges and best practices in software design in a global software development context. However, given the limited number of empirical papers to draw on, and lack of context information in the selected studies, we were unsure of how generalizable the findings were, plus we found some areas in which the challenges were left with no solution. Motivated by the need to add confidence to our results, and fill the gaps, we conducted our own empirical study. In the empirical study reported here we were careful to collect detailed information about the context of the software development (to include development process). Through several interviews with architects we gained visibility into the kind of challenges that they encountered in their day-to-day activities that included how they design and allocate tasks across their multi-site teams. We also asked each practitioner about how they tried to solve each of the challenges. In this way, we were able to fill some of the gaps identified in the SLR [17].

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).

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).

The dependencies in our conceptual model further illustrate the complex inter-relationships of challenges to practices and the holistic nature of architectural design in GSD, where the recommendation is to implement all practices to achieve the 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


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.


[45] K. Popović, Z. Hocenski, G. Martinovic, Do the software architects get the needed support for the job they perform?, in: Proceedings of the


