Lero Technical Report No.TR 2018 06 - Challenges and Recommended Practices for Software Architecting in Global Software Development

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

[a] *Tampere University of Technology, Laboratory of Pervasive Computing, Korkeakoulunkatu 1, P.O.Box 553, 33101 Tampere, Finland.*
[b] *Lero, the Irish Software Engineering Research Centre, University of Limerick, Ireland.*

Abstract

Context: Global software development (GSD), although now a norm in the software industry, carries with it enormous communication challenges. Careful management of task allocation across global development teams will alleviate some communication challenges. Deciding how the various development tasks are separated is the responsibility of the software architect, and occurs when the software is designed.
Objective: The current literature does not provide a cohesive picture of how the distributed nature of software development is taken into account during the design phase: what to avoid, and what works in practice. The objective of this paper is to gain an understanding of software architecting in the context of GSD, in order to develop a framework of challenges and solutions that can be applied in both research and practice.
Method: We conducted a systematic literature review (SLR) that synthesises (i) challenges GSD imposes on software architecture design, and (ii) recommended practices to alleviate these challenges.
Results: We produced a comprehensive set of guidelines for performing software architecture design in GSD based on 55 selected studies. Our framework comprises nine key challenges with 28 related concerns, and nine recommended practices, with 22 related concerns for software architecture design in GSD. These challenges and practices were mapped to a thematic conceptual model with the following concepts: Organization (Structure and Resources), Ways of Working (Architecture Knowledge Management, Change Management and Quality Management), Design Practices, Modularity and Task Allocation.
Conclusion: The synthesis of findings resulted in a thematic conceptual model of the problem area, a mapping of the key challenges to practices, and a concern framework providing concrete questions to aid the design process in a distributed setting. This is a first step in creating more concrete architecture design practices and guidelines.

Keywords: global software development, software architecture, software design, design practice, systematic literature review

1. Introduction

Global software development (GSD) can be defined as "software work undertaken at geographically separated locations across national boundaries in a coordinated fashion involving real time (synchronous) and asynchronous interaction" [1]. As companies are constantly seeking more ways to save on expenses, to expand and to find a larger skills pool, there has been an increase in the number becoming involved in GSD through outsourcing and the creation of development divisions in developing economies. With distribution comes distance in its many forms, which particularly affects communication and how employees are able to work together on the same task. Thus, allocating work in such a distributed setting becomes much more challenging when compared to collocated software development.

One of the key issues affecting how work is allocated to the distributed teams is the design of software architecture [3, 2]. The close structural dependency of the software architecture and the developing organization was already formulated as early as 1968 by Conway [4]: organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations. Conway's law has since been investigated in practice, most recently by, e.g., de Santana et al. [5], Bano et al. [6], and Imtiaz and Ikram [7], who all confirm that architecting practices follow Conway's law in GSD projects. However, researchers also raise the issues of how strict communication structures may actually isolate teams (that may appear closely linked otherwise) [6], and how the role of expertise should also be considered when allocating tasks [7].

In this study we conduct a review of the literature to

*Corresponding author
Email addresses: outi.sievi-korte@tut.fi (Outi Sievi-Korte), sarah.beecham@lero.ie (Sarah Beecham), ita.richardson@lero.ie (Ita Richardson)

examine whether it is necessary for architectural design to take into account the distributed nature of development work. The literature suggests that there are several differences in how software is developed in collocated teams, as opposed to distributed teams. However, the literature is unclear as to whether these differences impact design decisions.

Addressing organizational constraints - such as how communication is structured in reality – is especially crucial in GSD: the architecture should be such that it allows the sensible distribution of the implementation work to different persons, teams, and sites available for the project. In the optimal case, all the resources reserved for this work are utilized to their full potential [8], and the work is distributed in such a way that the communication overhead [9] caused by team distribution is minimized. However, such optimization is difficult to achieve due to the many challenges involved around distributed development. Furthermore, the level at which a software architect can actually consider all organizational factors varies due to demands set by functional requirements.

Thus, software architecture design is one of the most challenging tasks in the development of software systems [10]. In addition to considering organizational constraints, such as the aforementioned aspects, the architect's primary goal is to design a system that fulfills the given functional requirements. Furthermore, the architecture of a software system is the main vehicle to actualize the quality requirements of the system, thus directly dictating the quality properties of the product.

It is commonly acknowledged that best practices for software architecture design comprise loosely coupled components with well-defined interfaces [2]. Loose coupling follows ideas regarding modularization, information hiding, well-defined interfaces, and design by decisions (rather than functional sequence), given as recommendations for software design already during the 1970s by Parnas [11, 12, 13]. Parnas suggested that these practices aid in achieving software that is easier for new developers to understand and existing developers to modify and extend reducing the need to communicate with each other.

The assumption is that making components as independently implementable as possible would reduce the need for inter-team communication, and thus ease GSD related challenges. But is it that simple? Does practice follow theory in actual cases? In an empirical study conducted by Mustapic et al. [14], practitioners recognize the need to tailor their architectural design practices to take into account the complexity of implementing software across different geographic locations. Participants in this study warn: "We have seen examples of distributed development not being taken into account, this resulting in less than optimal architectural support for the process." Furthermore, Kwan et al. [15] show that the communication structure is more linked to the task structure of the system than to the actual component structure of the architecture - if tasks span across many components, are the dependencies between tasks recognized early enough in GSD?

So, how should the architectural design take into account the distributed nature of the development work? What kind of practices exist that can support distributed development? What are the GSD related challenges that are not as obvious as work allocation which need to be addressed while architecting software? Our goal in this study is to present answers to these questions by performing a systematic literature review (SLR).

This paper is organized as follows. In Section 2 we present background on GSD and software architecture as well as a brief overview of the related literature which highlights the need for our SLR. Our review methodology is presented in Section 3. The results of the SLR are given in Section 4, and in Section 5 we discuss the findings and provide practical checklists. Finally, conclusions and future work are given in Section 6.

2. Background and Related Work

In this Section we will cover relevant background on the topics of this study, namely global software development (GSD): its drivers and challenges and particularly the role of software architecture in a GSD context. We then look more deeply into software architecture and particularly its design, and finally, we discuss related work regarding research done in designing software architecture in GSD. Thus, this section focuses on the problem in GSD architectural design, leading us to construct our research questions that drive our SLR.

2.1. Global Software Development

Global software development has grown from a phenomenon to a paradigm in the past ten years. The main reasons behind distributing work to different sites across the globe are cost savings, access to larger skills pool, reduced time to market and proximity to customer. Other benefits such as shared best practices and innovations and improved task modularization also accrue [16]. However, there are many challenges within GSD that prohibit companies from realizing these expected benefits. Furthermore, a systematic review revealed that a clear majority of empirical studies in GSD are actually problem reports [17] rather than success stories. Based on the study by Ó Conchúir et al. [18], the expectations that shared best practices would spread and increase innovation appear mythical. Also, they note that many of the most common expected benefits have only been partially realized. For example, an expected gain from GSD is utilizing time zone differences to increase speed and reduce time to market. However, while some positive results have been achieved using well thought out processes [19, 20, 21, 22], Ó Conchúir et al. [18] found that these benefits are often mythical.

The reasons behind poor outcomes of distributed projects are largely due to the many challenges brought by the

three dimensions of distance - temporal distance (different time zones), geographical distance (physical distance between people), and socio-cultural distance (different ways of working, language, interpretation issues, etc.) [23, 24]. The challenges that distance brings are mainly due to increased effort required for communication, coordination and control [24]. For example, consider temporal distance. Due to time-zone differences, communication is in many cases asynchronous. Emails and even instant messages get answered with delay, and phone calls may not be made during office hours due to non-overlap of office hours between the sites. This results in delays in product development, rather than being able to effectively utilize the time zone difference to the company's advantage – the so called follow-the-sun method (FTS) [25]. If correctly managed processes are in place, FTS may be a solution in speeding the development, but coordination of the project requires specific practices developed particularly for FTS [20, 21, 22]. Furthermore, coordination, to support the division of work, is challenging due to possible delays, integration issues and missing skills. The dependencies between tasks need to be considered even more carefully than in a collocated project.

The challenges brought about by global distance are well recognized and documented in the GSD literature; as are the associated solutions to the distance problem. Distance mostly affects the soft aspects regarding teams and project management, and the proposed solutions often consider practices and processes rather than technical aspects [26, 27]. Research preference towards project/ engineering management is also clearly seen in tertiary studies of literature reviews of the field [28, 8]. Suggested practices include increasing traveling between sites, choosing sites in culturally similar locations, having a working infrastructure, and having a face-to-face kick-off of the project [19, 23, 29, 26]. On a larger scale, one success factor is simply that management must correctly identify the reasons for entering distributed development in the first place. A review showed that, in most cases, the reasons for going into GSD were actually unclear or irrelevant to the project [17]. Other studies showed that a company's success could be achieved when changes are based on the company's actual needs, and when methodologies and technologies used are given due consideration [30, 31].

The role of (software) product architecture is rarely brought to the fore when discussing challenges in GSD. Nonetheless, it is the main vehicle for determining the task structure, which is especially critical in GSD. The architect's key role in GSD is noted by Herbsleb [32], which was supported later by Britto et al. [33]. Herbsleb discusses how architectural design affects coordination and guides developers. He points out that software architects do not only design the structure of the product, but they also design the task dependencies among the teams designing and building the system. These dependencies have a straightforward effect on required communication between these teams. Architects should, in fact, be able to measure how well the given design fits the organization who are developing it. Similarly, Noll et al. [26] point out the need for a modular product architecture in order to reduce communication overhead. Babar and Lescher [30] also raise software architecture as a key strategy for success in a GSD project, as the architecture, and particularly how it answers to quality demands, and determines dependencies.

2.2. Software Architecture

Software architecture is defined by the ISO/IEC/IEEE 42010:2011 standard [34] as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution". Thus, software architecture design involves the definition of such elements and their relationships based on certain agreed principles. These elements are then given in a software architecture description, which essentially documents the decisions and the elements making the architecture.

Software architecture design was originally fairly unstandardized, and the need for quality architecture design was not recognized until closer to the 21st century [35]. For example, architecture descriptions took time to evolve from simple boxes and lines to more complex and standardized formats [35]. Ever since the boxes and lines era, the two key concepts in software architecture have been the separation of the system to entities (the boxes) that fulfill the functional requirements of a system, and the interaction (lines) between these entities. Moving on to component-based design in 2000, it has become increasingly common that at least one architecture view is made where the system is given in terms of components and the relationships between them [36]. Usage relationships are particularly important to describe, as they dictate the need for interfaces - what parts of a components functionality are made available for others to use? Separating tasks into components and having well-defined interfaces to increase modularization [11, 12, 13] would allow developers to work more independently, decreasing the need for communication between teams developing different components. To some extent, this follows Conway's law [4] in which software architecture design should reflect the communication structures of the organization. Herbsleb and Grinter [37], 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." If the architecture is not designed with the organization in mind, then it is natural that the organizational structure will be modified to suit the architecture (as suggested by recent work [38]), as the component dependencies will greatly dictate communication needs between different teams.

The most common practices and recommended solutions for certain recurring problems in architecture design

have been catalogued in various sources and forms. Software architecture styles, for instance, describe very high-level choices on how a system is to be implemented [39]. A software architecture can take various forms, for example, a layered or a pipeline typed structure, and interactions can be, for example, message-based [42] or service-based, implemented following the principles of SOA(P) [40], the RESTful approach [41], or applying the most recent trend of chopping functions to so-called microservices [43, 44]. The microservice approach has particularly evolved from the demands of increasing use of cloud computing [45, 46] and following *aaS (as a Service) approaches, where there is no single composition of software for one particular (group of) user(s), but flexibility is a must.

Common approaches are also product line architectures [10] and framework type systems [47], which enable large reuse of existing structures with only a limited number of exchangeable components to tailor the system to each customers specific needs. Lower-level principles called design patterns have also been recorded for different types of systems (see, e.g., Gamma et al. [48], Hohpe et al. [49], Schmidt et al. [50]), and architecture design can even be pattern-driven [51].

No matter what solutions are chosen, there is always a decision behind that choice. Software architecture design has, in fact, recently been focusing more on the decisions and the relationships between them [44, 52]. In a decision-centered approach, intangible concerns, such as what programming language or messaging protocol to use, or what kind of open source components to select, can be recorded and their relationships and effect on the code be tracked more easily. New approaches to software architecture reviews or evaluations are targeted more on such decisions, rather than expecting low-level component diagrams of a system [53]. Decisions made during the architecture design are also closely tied to the skills that are required to implement the architecture. Resources are a significant driving force in architecture design - if there is no expertise in certain technologies or solutions, they are unlikely to be selected or used as the primary solution, even if they would be the best solution if the skill was available.

Finally, architecture design decisions, driven by concerns, are evaluated against the non-functional requirements of a system. These are mostly associated with quality attributes [54]. In other words, a software system can be implemented in many ways to fulfill its functional requirements, but in order to fulfill the quality requirements, skillful architecture design is critical. Architectural choices dictate to a wide extent how the system responds to quality requirements. There are several quality-driven design approaches, one of the most notable ones being quality attribute-oriented software architecture design method (QASAR) as presented by Bosch [10]. Good software architecture design is thus essential to achieve good quality software. In order to achieve quality, the architect needs to consider both traditional non-functional requirements, such as measuring up to quality requirements in terms of modifiability, maintainability, performance, and how well the current resources support the chosen design decisions. If the available skills do not match the architectural choices, the resulting task will be overwhelming for the developers and require much more effort than estimated [9].

2.3. Architectural Design in GSD

Based on existing research in GSD and the common principles for software architecture design, we identified certain gaps in the literature. Firstly, it is unclear how quality demands are met in a globally distributed project. Quality assurance is difficult even in a collocated project, and is highly dependent on software architecture design. When communication is hindered, it is unclear how all quality demands are considered and how closely implementation follows the design. Jimenez et al. [31] point out the need for extra quality processes in GSD projects and Vaikar et al. [55] give a four-pillars approach which includes quality evaluation, but other than that no answers seem to exist. Second, is the issue of modularity - a driver behind task separation and how Conway's law appears in practice. We need to know the level of modularity which is considered in today's software development, and does it consider the global distribution of developers? Thirdly, are skills taken into account when designing software for distributed development? Taking into account resources and matching architecture to skills would reduce defects, and in a distributed setting there are fewer opportunities to lean on team members to teach less familiar techniques. Finally, does Conway's law go beyond the composition and structure of the architecture, or can it extend to technical decisions and skills required from the organizational perspectives? All the aforementioned questions relate to the process and practices of designing software architecture, and answers can be found by seeking recommended practices for architecting in a distributed organization. Additionally, there is a follow-up question: are there particular challenges in designing software architecture for GSD that might hinder the best possible practices?

There have been several systematic literature reviews in the area of GSD, as revealed by the tertiary study by Verner et al. [8]. Based on this study, it can clearly be seen that organizational factors, engineering or development process, and software project management issues are the most studied areas in GSD. Notably, from the listed 24 SLR studies, only one which involves design is listed: a review concentrating on architecture knowledge management (AKM) issues by Ali et al. [56]. Several studies consider software construction [57], but from the process viewpoint. This strongly suggests that there is a gap in design related research within GSD.

Ali et al. [56] 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 process of making architecture design decisions and other activities during the design/implementation phase in software development, 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.

Software architecture related studies in the context of GSD have also been reviewed by Mishra and Mishra [58]. They have identified that the main focal areas concerning architecting are knowledge management, process and quality, and framework and tool support. Again, this research does not capture the concrete practices of how to produce the architecture and achieve, for example, the quality goals.

By identifying aforementioned gaps in the literature we have constructed our research questions:

RQ1: What are the challenges in software architecture design when developing software across globally distributed sites?

RQ2: What are the recommended practices for software architecture design when developing software across globally distributed sites?

To answer these, we have undertaken the systematic literature review presented in this paper. Individual articles discussing software architecture in the context of GSD appear as part of the review, and are presented in the following sections.

3. Review Method

3.1. Protocol

For our review protocol we followed the steps recommended by Kitchenham [59] and Wohlin et al. [60]. The steps were derived for software engineering following the protocol as reported by Beecham et al. [61]. The protocol followed in this SLR is included in Appendix A. We used the following inclusion and exclusion criteria for selecting papers:

1. Include only studies concerning 2 or more sites in different geographic locations;
2. Include only studies concerning the design, development or implementation phase;
3. Include only papers with projects within the same company (offshore insourcing only, no offshore outsourcing[1]);
4. Exclude open source studies;
5. Exclude experimentations with student teams;
6. Exclude repeated studies (studies with two versions e.g., conference and journal);
7. Include only peer-reviewed studies;

Criteria 1 and 2 come directly from our research question - architecture design practices in distributed software development. While screening the abstracts of the studies we also considered architecture evaluation as design, as it can be undertaken during design formation. Whether evaluation-related studies were actually included depends on the content, where we ask "do they provide design practices"? Criterion 3 is to limit the scope and refine the question - in cases of outsourcing there are often quite separated teams for different tasks and less work undertaken on the same issues between sites. Further, we wanted to concentrate on the case where all sites belong to the same organization and have same organizational and business drivers behind the design. For this reason we also excluded open source projects (Criterion 4) and student projects (Criterion 5). In Criterion 6, in order not to skew results, we chose the most comprehensive version where a study had been duplicated, which is usually the journal version. Finally, we did not make exclusions based on the type of study, as long as it was peer-reviewed and had novel input, (Criterion 7) as challenges are often described in experience reports and reviews, rather than in pure research papers.

We searched six databases, and altogether we found 1820 papers in our searches. The search process and how many papers were found or selected in each phase is presented in Fig. 1. As a result of the process, a set of 55 reviewed papers are included in this study. Each stage of the selection process was validated by one or two independent researchers. Validation at each stage helped to sharpen the inclusion criteria. Validation is explained in more detail in Appendix A.

3.2. Sensitivity Analysis Results

When performing a systematic literature review one must be aware of skewed results, particularly if studies from very similar background dominate the selection. Results may be biased if the studies focus on a particular era, are performed by a certain research group(s) or in a small region [59], [60]. In order to identify potential bias in the selected studies we performed sensitivity analysis, where we analyzed the distribution of primary studies based on publication year, geography, type of study and publication venue.

The distribution of selected studies per type is illustrated in Fig. 2. A majority of the studies are case studies and experience reports which have a practical approach to the research questions. Experiment type studies mainly propose new approaches with initial validation, and their main contribution, as with reviews, is to gather the known

[1] Offshore insourcing: Leveraging company-internal resources situated in a different country
Offshore outsourcing: Leveraging external third-party resources situated in a different country [62]

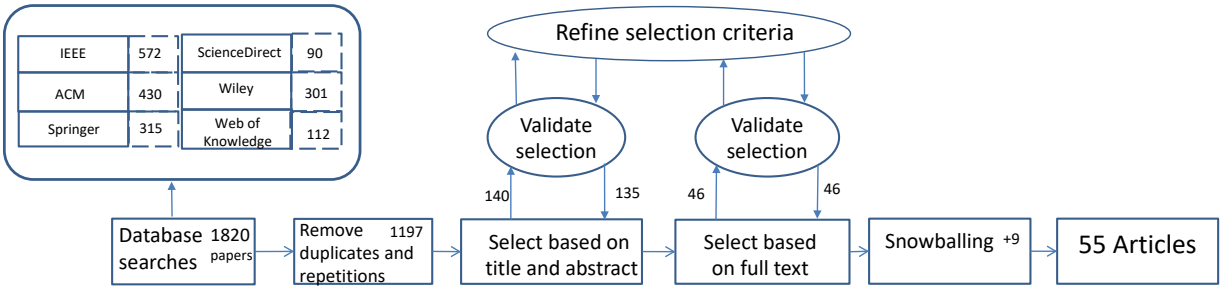

Figure 1: Selection process

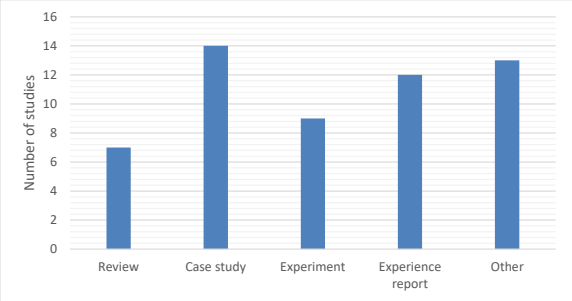

Figure 2: Publications per type

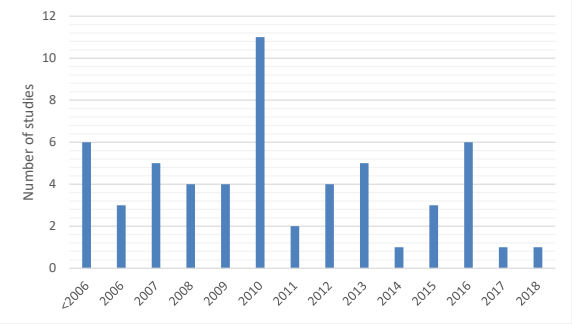

Figure 3: Publications per year

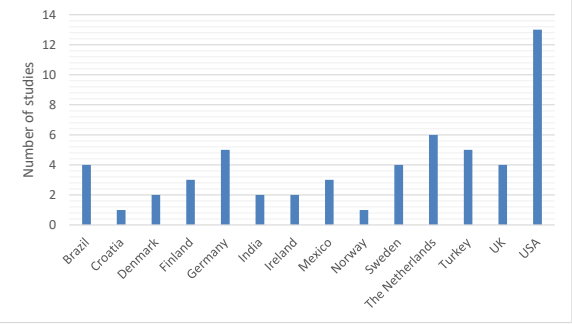

Figure 4: Countries represented in the studies

challenges. The studies of type "other" are multi-method studies, proposed tools, frameworks and position papers.

The distribution of publications per year is given in Fig. 3. We did not restrict our search to specific years, and the earliest publication included is from year 1999,

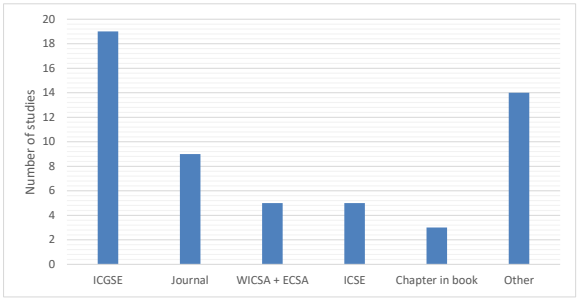

Figure 5: Publication venues

with a total of six studies published by 2005, after which the number of studies increases significantly. There is a distinctive peak in publications in year 2010, for which we could not find a clear explanation, other than we could see that research gains more interest every few years, then drops and starts building up again. After research in the area gained momentum again until 2016, we only found two publications in total from January 2017 to May 2018 that fit our selection criteria.

Naturally, many studies involve several countries due to the topic of our SLR, but the countries listed in Fig. 4 are the ones where the research group reporting the study resides. There are 14 countries in total, and almost 24% of the studies originate from the USA. Furthermore, apart from 4 studies originating from Brazil, three originating from Mexico and two from India; all other studies are performed by researchers in Europe (60%).

Finally, Fig. 5 shows where primary studies were published. A majority of the studies were published in the proceedings of the International Conference on Global Software Engineering (ICGSE). Several studies were published in conferences focused on architecture, Working International Conference on Software Architecture (WICSA) or European Conference in Software Architecture (ECSA), and in the largest general software engineering conference - International Conference on Software Engineering (ICSE). Nine articles were published in journals - four of them in IEEE Software, while all others were in different journals. The column Other represents here the 13 different conferences where 14 of the studies were published. The publication venues were all clearly software engineering oriented.

We additionally analyzed the distribution of primary

studies based on what kind of software development process was used, what was the domain under study, and what size was the company in case. We found that in 71% of the studies the used software development process was not specified. In 15% the case study company used some form of agile process, 5% followed the waterfall process, and in 5% the study described several cases, where one or more used agile and one ore more some other process. The company size was undefined in 69% of the studies, while in 29% the company was multinational or large. In 62% of the cases the domain was not specified, while 13% of the studies covered cases from various domains. The largest individual domain was telecommunications, which was represented in 7% of the studies.

Therefore, this sensitivity analysis highlights a bias in the studies that we need to be aware of in our data synthesis. Themes found are likely to take on a mainly Western view of the problem area. Further, we cannot be sure on how much of the reported practices depend on used process or size of company, as in over two thirds of the studies this background information was not specified. On a more positive note, there is a good spread of methods used across all studies which should produce a rich set of data that we can draw on to answer our research questions.

3.3. Data Synthesis

Because of the versatility of the material, we decided to synthesize the data with thematic synthesis [63]. Thematic synthesis organizes the material in a way that makes it easier to identify the key findings of the primary studies. Using thematic synthesis enables a rather rich presentation of data within limited space and is more flexible [64, 65]. It is also particularly suitable for studies with mixed methods [60], as is the case here. There are weaknesses as well (mostly due to the transparency of the process), but both the strengths and the applicability of this synthesis technique made it the best choice for this study.

Our thematic synthesis is inductive and data driven, i.e., we have not tried to fit the findings into an existing theoretical framework, but the themes have been produced purely from the data itself. We followed the procedure as described by Braun and Clarke [63], who define the following six phases for performing thematic analysis.

Phase 1: familiarizing oneself with the data. This was done during the review process, as all included primary studies were fully read by at least one author, while others read a portion of the studies as part of the validation process.

Phase 2: generating initial codes. We generated the initial codes by collecting extracts from the primary studies that would directly answer the research questions. From the extracts we could identify a description of a challenge or some kind of practice, technique or advice on how to perform architectural design. We then developed the actual codes by identifying common or related words from the extracts.

Phase 3: searching for themes. In this phase the codes were sorted into and under potential themes. The codes were analyzed and we considered how different codes could be combined to form a more general, overarching theme. The codes were arranged based on their relationships to each other, resulting in sub-themes under the main themes.

Phase 4: reviewing themes. After forming a set of candidate themes, they were refined. We had two iterations of reviewing the themes - firstly, we re-read the extracts that were placed under each theme, and confirmed whether it still belonged under the suggested theme, or moved it under another theme. Secondly, again, as proposed by Braun and Clarke [63], there was a discussion among the researchers about the themes, which lead to a significant rewording of the theme names and theme re-organization.

Phase 5: defining and reviewing the themes. This phase identified the "story" behind a theme, conducting and writing an analysis of what each theme reveals about the subject.

Phase 6: writing the report.

After performing these steps, we had identified a set of challenges and recommended practices related to architecture design in GSD, grouped under themes. Through analysis of the challenges and good practices we were able to elicit relationships between the themes. This brought out a thematic conceptual model of the problem field - architecting in GSD. Furthermore, based on the analysis performed, we mapped the challenges and practices into a concern framework. The model and concern framework are presented in the Results section.

4. Results

We will go through the results of our analysis by first revisiting the research questions to summarize our findings. We will then proceed by presenting a conceptual model derived from the literature in Subsection 4.2. Finally, in Subsection 4.3 we will present challenges and practices for architecting in GSD in more detail, including how each challenge and solution maps to the research papers (Tables 1-8) and by extension to the research questions.

4.1. Revisiting the Research Questions

RQ1: What are the main challenges in software architecture design when developing software across globally distributed sites?

From the analysis it became quite clear that the main challenges stem from the distributed nature of GSD, particularly the organization and the teams. Challenges include considering organization structure in the design, finding the issues that affect how distributed teams can best work, and managing awareness in distributed teams - including awareness of architecting practices and guidelines. Defining clear ways of working, including quality and change management practices is particularly important to ensure

that responsibilities regarding these issues are clear and that they are handled with due diligence.

RQ2: What are the recommended practices for software architecture design when developing software across geographically distributed sites?

The discovered recommended practices include the need for a well-thought out work distribution that mirrors the structure of the product and the structure of the organization. Work should be broken into manageable pieces. Furthermore, having clearly defined design practices and interfaces will enhance loose coupling and support the aims of work distribution. The need for communicating the architecture across different sites should be recognized, and different views used as needed. The distributed nature of the organization should be considered when assigning architects and organizing the design work.

4.2. Architecting in GSD

Our thematic analysis of the literature allowed us to produce a conceptual model of the problem area, shown in class diagram format in Fig. 6. Themes (concepts) are presented as classes, practices and challenges are given (in condensed form due to space restrictions) as class members (coded with P1-P9 for practices and C1-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 special relationship between themes and directly derived sub-themes. We have also included a note on Conway's law to clarify the relationships between Organization, Task Allocation and Design Decisions. Note that these are only the themes that arose from our SLR. While the concepts as such are familiar to the architecting community, the given themes are ones that appear to have particular importance in the context of GSD.

As is commonly known, the core of architecting is the Design Process, and this applies to GSD as well. The design process is often managed and dictated by the architect, or a team of architects, who has a certain role. Another option is to have a team of developers sharing architectural responsibilities, as may be in the case where agile practices are used. The role of architect is not self-evident, and contains many social and organizational aspects in addition to holding the main responsibility for the design decisions [44] and the architect's managerial role is particularly emphasized in GSD.

The design process thus entails *project management* issues as well as *design decisions*. We are separating these into two separate concepts, as often different roles are needed to take responsibility for each. Design decisions and project management are also often conflicted - the best possible design in terms of quality may not always be possible due to project management restrictions, and critical design needs may, in turn, result in resource acquisition or organization restructuring. These two core concepts of architecting are noted with stereotypes in Fig. 6 to distinguish under which core concept each theme falls. Further, classes where these concepts overlap are marked with a special stereotype "Design decisions and Project Management". The core concepts overlap most clearly in Ways of Working. Ways of Working include, for example, processes to check compliance to requirements, quality management, and communication, which are clearly project management tasks. However, all these tasks may have a huge influence on design decisions, and it is impossible to completely separate design-related aspects from managerial aspects in this area. For example, a manager decides the processes for Architectural Knowledge Management (AKM), including how architectural artefacts are stored. However, the actual artefacts to be stored are a direct result of design decisions. Furthermore, design decisions make up the architecture and thus greatly affect its understandability, but how the architecture is presented and communicated may have just as big an effect, and these are determined by Ways of Working. In Figure 6 we have given three sub-themes for Ways of Working - AKM, Quality Management and Change Management. There is a strong link also between Change Management and Modularity. A modular design will help in managing changes, as it is possible to isolate software changes to well-defined "modules". However, poor change management will make maintaining a modular architecture more difficult.

Design Decisions follow common Design Practices. Most commonly accepted design practices, in turn, strive for a modular architecture. How well Modularity has been achieved can be calculated by using the metrics for Coupling (dependencies between components).

In addition to Ways of Working, project management and design decisions also overlap in Task Allocation. At the heart of task allocation is Conway's law - the software architecture and its developing organization will end up mirroring each other. Either the organization will be a driver in separating the architecture into components that fit the organization, or the software architecture will drive the restructuring of the organization to fit the development tasks' required communication structure. Thus, both the design and particularly the modularity of the architecture and the organization (its structure and resources) determine task allocation. Furthermore, task allocation, through Conway's law, also affects the design and the organization in turn.

In addition to themes and how they relate to each other, we show the challenges [C1-C9] and practices [P1-P9] we found in our analysis under each theme in Fig. 6 – these challenges and practices effectively answer our research questions, as challenges are findings for RQ1 and

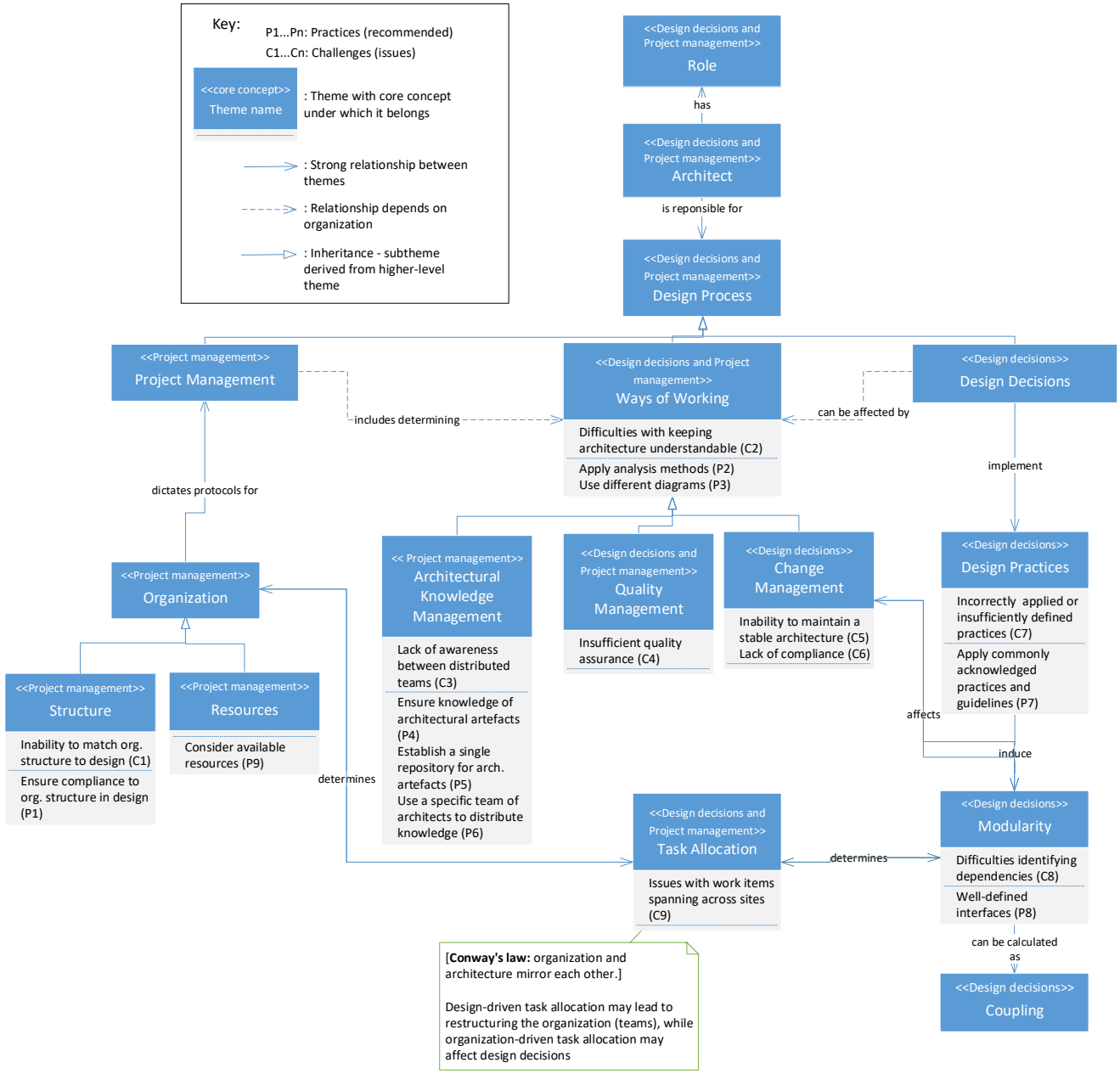

Figure 6: Architecting in GSD.

practices for RQ2. Individual studies often only discussed either challenges or practices to architecting – it was rare that actual solutions were proposed to perceived challenges. Using the conceptual model as given in Fig. 6 we were able to match challenges with the practices we found in the reviewed literature. In many cases we could see that the practices mirror the challenge.

4.3. Concern Framework

Having answered the RQs, we developed a concern framework from our SLR. The concern framework essentially maps individual practices to challenges, thus providing solutions to identified problems. Challenges and recommended practices are further broken down to concrete concerns, which are given at such a detailed level that a practitioner should be able to see if such concerns are addressed. We will go through the concern framework focusing on themes introduced in Fig. 6, discuss the challenges and practices in more detail and complement them with lower-level concerns which will aid in practical architecting. The next sub-sections consider challenges and practices for the Organization, Ways of Working, Architectural Knowledge Management, Quality Management, Change Management, Design Practices, Modularity, and Task Allocation. Citations for the 55 papers included in our literature review are given in numeric form with a

Table 1: Concerns for Organization

ID	Challenge/Practice	Concerns	References
C1	**Challenge** - Inability to match organization structure and practices to architectural design and assign tasks accordingly	Difficulties with making the organization reporting structure match the geographic distribution of tasks (C1_co1)	[SLR1],[SLR2], [SLR3],[SLR4]
		Overlooking organization management (C1_co2)	[SLR5],[SLR6], [SLR7]
		Insufficient matching of code to available resources (C1_co3)	[SLR2]
		Lack of alignment between architectural decisions to organization structure and not reflecting architectural changes to organization (C1_co4)	[SLR1],[SLR8]. [SLR9],[SLR10], [SLR11],[SLR12]
		Difficulties with correctly identifying dependencies between work units and thus assigning work to distributed teams (C1_co5)	[SLR13], [SLR14], [SLR15]
		Inability to maintain experts from all domains required for change implementation (C1_co6)	[SLR2], [SLR8]
		Misaligned interests and undesirability of tasks make task distribution challenging (C1_co7)	[SLR16]
P1	**Practice** - Ensure that organization/work allocation is compliant with architecture design	Ensure that components that will be dispersed to distributed teams are loosely coupled or otherwise plan component breakdown to independent modules based on distribution of teams (P1_co1)	[SLR1],[SLR4], [SLR9],[SLR13], [SLR14],[SLR17], [SLR18],[SLR19], [SLR20],[SLR21], [SLR22]
		Retain tightly coupled work items at one site (P1_co2)	[SLR2],[SLR4], [SLR13],[SLR14], [SLR15]
		Let the architecture determine how tasks are allocated, and who is responsible for each task's teams (P1_co3)	[SLR4],[SLR23], [SLR24]
		Break work items to easily manageable pieces (considers one subsystem, can be handled by one person) (P1_co4)	[SLR2], [SLR21]
P9	**Practice** - Consider available resources from different sites in the design	Identify where the domain expertise lies and allocate tasks accordingly (P9_co1)	[SLR2],[SLR5], [SLR19],[SLR20]

bibliography at the end of this paper in a separate section ("Selected 55 Papers for the SLR"). Note that citations within the text from papers selected for the SLR are noted with the prefix "SLR".

4.3.1. Organization

Table 1 shows the challenges and recommended practices specific to the organization, its structure and resources. Each challenge and practice is given an ID (P for Practice, C for Challenge and a number). Challenges and practices are further broken down to concerns - smaller scale challenges or more detailed ways to implement the practice.

The challenge regarding the organization relates to matching the organizational structure and practices to architectural design and task assignment (C1). This challenge already explicitly shows how tightly linked the organization and architectural design are and how they overlap in allocating tasks. We have grouped here the challenges and

Table 2: Concerns for Ways of Working

ID	Challenge/Practice	Concerns	References
C2	**Challenge** - Difficulties in keeping architecture decisions understandable	Insufficient understanding of architectural decisions in teams and other stakeholder groups (C2_co1)	[SLR3],[SLR4], [SLR12],[SLR14], [SLR21],[SLR25]
P2	**Practice** - Apply analysis methods for detecting dependencies and conflicts	Use (call) graphs/matrices to depict and detect coupling (P2_co1)	[SLR9],[SLR13], [SLR18],[SLR20], [SLR25],[SLR27]
		Use visualization of decisions/metrics (P2_co2)	[SLR11],[SLR25], [SLR28]
		Use collaborative modeling (P2_co3)	[SLR29]
P3	**Practice** -Use different types of diagrams for different stakeholders	Use a variety of diagrams to promote awareness (P3_co1)	[SLR4],[SLR12], [SLR14],[SLR25], [SLR30],[SLR31], [SLR32],[SLR33]
		Don't over-rely on UML diagrams (P3_co2)	[SLR30]

practices that consider the organization prior to allocating tasks, i.e., what should be considered when task allocation is at planning stage, where the focus is on fitting the organizational practices, structure and resources to design – task allocation challenges being a result of not solving this organizational challenge and related concerns (C1_co1, C1_co2, C1_co3, C1_co4 and C1_co7). Concerns also touch other themes, such as implementing changes (C1_co6) and identifying dependencies (CI_co5), further showcasing how tightly linked the design practices and project management issues are when architecting.

The corresponding practices state that one should ensure that the architectural design and organization are a match (P1) and remember to consider the available resources (P9), especially domain expertise (P9_co1). The concerns related to structure provide more concrete guidelines - ensuring that components sent to different sites are loosely coupled ones (P1_co1), keeping tightly coupled items to one site (P1_co2), using an architecture-driven approach for task allocation (P1_co3) and breaking down work items to manageable pieces (P1_co4).

One can clearly see that organization related practices seem to follow Conway's law – but which comes first, the organization or the architecture, and how to match them? Matching, in turn, boils down to task allocation, since the underlying cause for all concerns here is the need for developers to communicate when they are working on dependent components, and distance makes communication more difficult.

4.3.2. Ways of Working

Challenges and recommended practices related to Ways of Working are listed in Table 2. As seen in Figure 6, the Ways of Working theme considers many aspects of the design process where project management and design decisions overlap. There are three sub-themes for Ways of Working, which will be covered separately. On this level we identified a challenge in keeping the architecture decisions understandable (C2), with it's related concern (C2_co1) of decisions not being understood by developers or other stakeholders who should be able to understand the architecture. This could be aided by applying analysis methods (P2), particularly using different methods to detect (P2_co1) and visualize (P2_co2) different aspects of the architecture. Furthermore, stakeholders have different backgrounds, which should be taken into account when introducing the architecture, and understandability can be aided by using different views (P3) and selecting the view based on the stakeholder's needs and level or expertise on the design (P3_co1). Further, to aid understandability, architects should not over-rely on technical diagrams, such as UML, (P3_co2) as evidence shows that different sites may have difficulties interpreting such diagrams due to different backgrounds.

4.3.3. Architectural Knowledge Management

One essential sub-theme under Ways of Working is Architectural Knowledge Management (AKM). AKM is a sensitivity point between project management and design - architectural knowledge is key for quality design, and communicating that knowledge is key for successful implementation. AKM related challenges and recommended practices are presented in Table 3. The main challenge in AKM is indeed lack of awareness, i.e., failure to communicate (C3). The more concrete concerns highlight specific issues around communication, such as not using an electronic knowledge management system even if there is one available (C3_co4), and lack of communication leading to

Table 3: Concerns for AKM

ID	Challenge/Practice	Concerns	References
C3	**Challenge** - Lack of awareness between distributed teams and problems due to communication and knowledge management challenges	Problems caused due to not involving an architect with sufficient technical background knowledge (C3_co1)	[SLR4],[SLR18]
		Difficulties in effective creation and sharing of architectural artefacts (C3_co2)	[SLR12],[SLR14], [SLR15],[SLR16], [SLR34],[SLR35]
		Difficulties in maintaining a common view of the project (C3_co3)	[SLR4],[SLR12], [SLR14],[SLR36], [SLR37]
		Inconsistent usage of electronic systems for knowledge sharing due to preference of social networks (C3_co4)	[SLR35],[SLR36]
		Impractical condensing of knowledge due to high dependency on one lead architect (C3_co5)	[SLR17],[SLR37]
		Insufficient architectural documentation (C3_co6)	[SLR35],[SLR39], [SLR40],[SLR41]
P4	**Practice** - Ensure knowledge of architectural artefacts and practices across sites with communication	Communicate architectural artefacts and practices clearly to all sites (P4_co1)	[SLR3],[SLR4], [SLR8],[SLR14], [SLR17],[SLR21], [SLR25],[SLR31], [SLR35],[SLR37], [SLR41],[SLR42], [SLR43],[SLR44], [SLR45]
P5	**Practice** - Establish a single repository for architectural artefacts	Maintain a single repository for architectural artefacts accessible to all (P5_co1)	[SLR8],[SLR15], [SLR21],[SLR22], [SLR25],[SLR30], [SLR41],[SLR43], [SLR46]
P6	**Practice** - Use a team of architects to collect and distribute the knowledge	Define clear responsibilities for architecture team to handle changes that span through several components and/or sites (P6_co1)	[SLR8],[SLR9], [SLR17],[SLR31], [SLR36],[SLR37], [SLR38],[SLR47],
		Ensure each site has representative architect (P6_co2)	[SLR9],[SLR17], [SLR38]
		Arrange collocated activities for architecture team to promote awareness (P6_co3)	[SLR16],[SLR26], [SLR37],[SLR38], [SLR48]
		Establish a team of architects for handling communication between different stakeholders and teams (P6_co4)	[SLR25],[SLR37]

different stakeholders having different views of what the architecture actually is (C3_co3). The importance of recognizing the architect's role is also brought out in AKM - not having an architect with a technical background can

Table 4: Concerns for Quality Management

ID	Challenge/Practice	Concerns	References
C4	**Challenge** - Insufficient quality assurance	Delegating design decisions to local teams deteriorates quality (C4_co1)	[SLR4],[SLR9], [SLR22],[SLR46]
		Insufficient quality management (C4_co2)	[SLR9] [SLR12], [SLR14],[SLR21], [SLR25],[SLR34], [SLR48]
		Decentralized data and state management lead to inferior quality (C4_c3)	[SLR9]

Table 5: Concerns for Change Management

ID	Challenge/Practice	Concerns	References
C5	**Challenge** - Inability to maintain a stable architecture	Lack of stability in architecture leads to difficulties in applying design rules and dividing tasks (C5_co1)	[SLR18],[SLR24]
		Unclear ownership of architectural elements (C5_co2)	[SLR7][SLR44]
C6	**Challenge**-Unforeseen problems due to lack of compliance to organization, business process or architectural specification	Difficulties ensuring compliance of modular design throughout the lifecycle and changes in organization (C6_co1)	[SLR2],[SLR10]
		A lack of conformance to architectural specification (C6_co2)	[SLR11],[SLR30]
		A lack of compliance to the business process (C6_co3)	[SLR10],[SLR11]

lead to serious issues as the architect is unable to recognize and communicate potential problems in the design (C3_co1). The responsibility for creating and sharing architectural artefacts must be clear for such knowledge to be effectively distributed (C3_co2), as well as the level of detail required to understand the artefacts (C3_co6). Furthermore, if there is only one central architect in a distributed project holding all the information, distributing it to others is challenging (C3_co5).

As there have been a significant number of studies focusing particularly on AKM, there are also more working practices found than for other themes. First, studies simply remind the practitioner to communicate (P4 and more precisely P4_co1). Secondly, to solve some basic knowledge management issues, it is advised that in distributed projects there should be one single repository where all architectural artefacts are stored and all relevant stakeholders should have access to that repository (P5 and specifically P5_co1). The last practice, however, is not as self-evident - using a team of architects (instead of just one central architect) to collect and distribute knowledge (P6). More concretely, each site should have its own representative architect (P6_co2) and these architects should have collocated activities throughout the design process to increase awareness within the architect team (P6_co3). Finally, special concerns (P6_co1) are raised on clearly defining responsibilities for the architect team to handle changes that span across components or sites as well as using the team of architects as mediators, communicating between different stakeholders (P6_co4). As most practices concentrate on keeping tasks separated, it is especially important to recognize that pure separation is not always possible, and there should be a clear plan on how to handle those tasks that are likely to cause most problems.

4.3.4. Quality Management

A key driver in architecture design should be the quality requirements of the system. Thus, quality management should be defined when considering the ways of working in a design process. Table 4 lists the concerns in managing quality - no recommended practices were found under this theme. Quality is difficult to gain, measure and maintain, and unsurprisingly we found a challenge related to insufficient quality assurance (C4). There were reports on deteriorated quality due to design decisions being delegated to teams with no one checking the decisions until it was too

late (C4_co1). A more general concern was simply insufficient quality management processes (C4_co2) - in most cases no one was assigned the responsibility for quality assurance. Finally, a concrete concern was raised regarding decentralized data and state management and their effect on quality (C4_co3).

4.3.5. Change Management

Similar to quality management, change management is an important area of architecting, but there are no good practices found in the literature to aid change management challenges, as listed in Table 5. There are two main challenges - maintaining a stable architecture (C5) and lack of compliance (C6). An architecture may become unstable if ownership of architectural elements is unclear (C5_co2), and instability will, in turn, hinder applying design rules (deteriorating quality) making task allocation difficult, i.e., if architecture is unstable, so is the modular division (C5_co1). While maintaining stability has to do with change within the architecture, concerns have been raised regarding lack of compliance to aspects beyond the architect's control (changes within the organization, requirements, process, etc.) (C6_co1). In some cases such compliance was lacking already at the start of the design process (C6_co2, C6_co3).

4.3.6. Design Practices

At the core of architecting are design practices, and concerns for this theme are listed in Table 6. The challenge we found was caused by insufficiently or incorrectly defined or applied practices for design and development (C7). Clearly, problems arise if a) practices are not defined clearly enough or b) practices are not applied as they are defined. This is most emphasized by simply ignoring or incorrectly using agreements on design (C7_co5). This concern specially appears in GSD, as ignorance of such agreements is often a result of different working cultures and principles, and the reasoning behind rules may not be clear for remote sites. Insufficient interface specifications (C7_co3) is particularly challenging in GSD as well, as separation of tasks relies so heavily on separation of components, which in turn is achieved through interfaces. Having commonly agreed practices and well-defined interfaces is critical to enable a working product to be deployed from the separated tasks. These difficulties were found and described by Herbsleb and Grinter [SLR50]:

Following design agreements, teams developed each component at a single site in relative isolation from other sites' teams. Each team built simulators to represent other components that their code would need to interact with. Interface specifications lacked details, such as message type and assumptions on performance, and developers proceeded unknowingly with incorrect assumptions about other components. Because of simulators the discrepancies remained hidden for long time

A related concern discusses assumptions (C7_co4). Separated teams tend to assume what others are doing if agreements are insufficiently defined or there is a lack of communication. This again will often lead to component mismatch. More detailed concerns relate to prioritization rules and inconsistent versioning (C7_co1 and C7_co2). As a way to tend to this challenge, practitioners simply suggest applying commonly acknowledged architecting practices (P7). Such practices are listed by Sauer [SLR21]:

Stick to proven design and architecture principles: information hiding, loose coupling, strong cohesion, design by contract, open closed principle, avoidance of type interdependencies, etc. .. Principles gear towards understandable software with fewer dependencies, thus easing maintainability and further development.

More concretely, one should ensure that practices are well-defined (P7_co2), and there should also be well-defined principles for coding (P7_co1). This would ensure that the code actually matches the architecture. More detailed concerns relate to using a service oriented approach and using prototyping (P7_co3, P7_co4). Finally, an important aspect in fitting architecture to requirements is to consider business goals in design (P7_co5).

While our goal was to find concrete practices for architecting in GSD, the actual technical details given in the studies were rare. The most detailed approaches identified are the following:

- Well defined invocations, narrowly defined interfaces [SLR9], [SLR22], [SLR36]
- Week-long workshops for design phase [SLR16]
- Architectural styles [SLR18], [SLR45],[SLR52]
- Documenting architectural design decisions and storing them in a commonly available repository [SLR15], [SLR25], [SLR41]
- Keep a global repository of architectural components, including key aspects such as responsibilities, non-functional characteristics, assignment to layers and interfaces. [SLR22], [SLR25]
- Identifying logical [SLR24] and dynamic, timing and resource dependencies [SLR32]
- Detailed code conventions [SLR30], [SLR35]
- Work items that affect several subsystems are split into distinct modification requests (MR) so that each MR affects one subsystem. A work item in a subsystem that is too much for one person is organized into several MRs, each for one person. [SLR36]
- Application Programming Interfaces [SLR11]
- Service Oriented Architectures [SLR4], [SLR15], [SLR33], [SLR44]

Table 6: Concerns for Design Practice

ID	Challenge/Practice	Concerns	References
C7	**Challenge**-Multiple problems due to insufficiently or incorrectly defined or applied practices for design and development	Insufficient prioritization rules (C7_co1)	[SLR4], [SLR9]
		Inconsistent versioning (C7_co2)	[SLR9]
		Insufficient interface specifications (C7_co3)	[SLR1],[SLR4], [SLR50]
		Incorrect assumptions made during design (C7_c4)	[SLR50],[SLR51]
		Ignorance of or incorrect use of principles, rules and guidelines for architectural design and knowledge management (C7_co5)	[SLR9],[SLR11], [SLR14],[SLR34], [SLR36],[SLR44], [SLR45],[SLR51], [SLR52],[SLR53]
P7	**Practice** - Apply commonly acknowledged architecting practices and guidelines	Ensure that teams develop code based on common design agreements (P7_co1)	[SLR1],[SLR4], [SLR8],[SLR17], [SLR22],[SLR37], [SLR44],[SLR50]
		Use common architectural practices and ensure they are well-defined (P7_co2)	[SLR4],[SLR13], [SLR15],[SLR18], [SLR21],[SLR24], [SLR34],[SLR38], [SLR47],[SLR48], [SLR52],[SLR54]
		Use a service oriented approach (P7_co3)	[SLR15],[SLR20], [SLR44]
		Use prototyping (P7_co4)	[SLR30]
		Include business goals in design (P7_co5)	[SLR22],[SLR25]

- Microservices [SLR22]
- Continuous integration and deployment [SLR22]
- Wizards [SLR30]

We can see that the given detailed practices are mainly concerned with managing work items and enabling loosely coupled components, while there are also some mechanisms for how to achieve common design practices.

4.3.7. Modularity

A particular aspect of architecting is modularity, i.e., separating functionality into independent components or modules. This is particularly important in GSD, as modularity is often the driver for task allocation. Modularity related concerns are listed in Table 7. The challenge in achieving a modular architecture lies in correctly identifying architectural dependencies and decoupling components (C8). As the concerns reveal, not all dependencies are self-evident - in addition to operation calls, there are other code level decisions (e.g., common variables and object states) as well as resource dependencies (skills) that may introduce dependencies between components and the developers implementing them (C8_co1). Being able to identify all relevant dependencies requires skills, as demonstrated by Bass et al. [SLR9]:

Currently the view is that the technical mechanisms that cause task interdependencies are invocations across modules (assuming a module is a task assignment to a single team). This view leads to a relatively narrow focus when architects and managers attempt to align the architecture with the organization. We have found that this narrow view is not sufficient. There are additional architectural mechanisms such as state management, resource utilization, and schedule synchronization which can also require extensive interaction among teams.

Furthermore, the actual technical practice of decoupling should be carefully considered from all perspectives, as dos Santos and Werner [SLR28] point out:

Decoupling components may or may not decouple tasks. Adding an intermediary where the most difficult dependen-

Table 7: Concerns for Modularity

ID	Challenge/Practice	Concerns	References
C8	**Challenge**- Difficulties in identifying architectural dependencies and decoupling components to sufficiently separate work items	Insufficient decoupling, cross-component features (C8_co1)	[SLR1],[SLR17], [SLR34],[SLR37], [SLR45]
		Inability to recognize dependencies between or created by architectural decisions. (C8_co2)	[SLR4],[SLR9], [SLR14],[SLR24], [SLR29],[SLR51], [SLR54]
P8	**Practice** - Promote loose coupling with well-defined interfaces	Implement well-defined interfaces to increase modularization and aid loose coupling (P8_co1)	[SLR1],[SLR4], [SLR9],[SLR13], [SLR17],[SLR24], [SLR44]

Table 8: Concerns for Task Allocation

ID	Challenge/Practice	Concerns	References
C9	**Challenge**-Issues with work items spanning across several sites	Increased amount of effort with modifications involving several developers across different sites (C9_co1)	[SLR2]
		Increased needs for coordination when using experts from different sites (C9_co2)	[SLR2],[SLR8], [SLR37],[SLR55]

cies are semantic rather than syntactic may in fact make the task coordination problem harder. Understanding task coupling would, for example, let us predict if the need for coordination is so intense that teams need to be co-located.

Keeping features within one component also is not an easy task, and thus dependencies between components can creep into the architecture if cross-component features are easily accepted (C8_co1).

The related practice encourages well-defined interfaces to promote loose coupling (P8), in this context to increase modularity (P8_co1). Well-defined interfaces were already discussed in relation to Design Practices, but they are particularly important in achieving modularity in architecture.

4.3.8. Task Allocation

The last theme in our model is Task Allocation, specific concerns for which are given in Table 8. Task allocation is at the cutting point of design and project management - how to allocate tasks, considering the given resources, organization structure and design? What comes first - design or organization? Thus, task allocation related challenges and practices have already been covered partially in relation to organization, design practices and modularity. We identified one additional challenge. This relates to when tasks have already been allocated to span across several sites (C9). One should be aware that, in this case, there will most likely be an increased amount of effort as developers from different sites need to communicate in order to complete the task (C9_co1). This will lead to increased needs for coordination (C9_co2). Based on the findings of our SLR, allocating tasks across sites should be avoided (unless the company is practicing FTS, which is a special case). The checklists provided in the Section 5.1 should help to avoid this particular challenge.

5. Discussion

An interesting point in our findings is the similarity between collocated and GSD projects and products, which is emphasized by Clerc [SLR43]:

In contrasting software products developed using GSD with products developed on a single development site, we did not find significant differences between the two groups.

The recommended practices and challenges we discovered are largely common also in collocated projects, but there appears to be different emphasis in GSD due to global distance. As collocated projects do not experience the challenges brought by the three dimensions of global

distance, for example task allocation would not be such an important issue. Hence, modularity would likely be also less emphasized in collocated projects (although good architects strive for it anyway).

However, we also found recommendations that the distributed nature of the development work should be taken into account in the design, and from the SLR we were able to identify recommended practices targeted specifically for distributed software design. Further, previous research by Casey and Richardson [25], indicates that not taking the distributed nature of development into account causes problems – "too often the implementation of an outsourcing or offshoring strategy has been seen as simply the replication of those strategies which are implemented for collocated software development."

Thus, it could be argued that the difference between distributed and collocated architecting stems from a requirement to adapt the design practices to consider global distance by, e.g., having a multi-sited team of architects or leveraging local knowledge from different sites. Whether this need is widely recognized in practice is not self-evident. In any case, not recognizing the need for different processes when working in a distributed environment is a cause for concern.

As for actual distributed architecting – an overview of the identified challenges and recommended practices presented in Tables 1-8 clearly reveals a key issue in GSD: the most acknowledged practices relate to the most commonly identified challenges. However, practices are still rather abstract, and scarcely applied. As an example, Clerc reported [SLR43]:

Decomposition and layering and design patterns are well-known approaches to reduce complexity but application is scarce both in collocated and GSD products.

Furthermore, many challenges and practices seem to even mirror each other, such as C1 (Inability to match architecture to organization) and P1 (Ensure architecture and organization match). On such a general level, practices are not very helpful. This, however, is the level at which practices are often reported. To aid the practical design process, our concern framework was designed to bring practices to a more concrete level.

5.1. Checklists

Taking the concerns a step closer to practice we present checklists for the practitioner. The checklists are given in table form with a question, rationale behind the question and an example answer (see Tables I-VII in Appendix B). The questions and rationale are based on the primary studies of the SLR. Questions were formulated by examining the high level analysis of the data synthesis (themes) together with the original text extracts and reworking the data into question format. The theme gave a context into which the extract was tied. Focusing on a detailed extract allowed us to pinpoint very concrete issues, which, in turn, could be reworked into straightforward questions. To clarify this process, we will present an example from the checklist for AKM (Table III). Under this theme, we had three quotes from the same source [SLR17]:

- "Case study GLOembed applied a model in which only architects between between the teams communicated with each other an a lead architect travelled to all the different team sites to communicate a bout the strategic plans and road maps"
- "All these teams were dependent on a central and top-down unit for inter-team coordination, which implied challenges in .. coordination needed for integration, and high dependency on one lead architect"
- "Teams were geographically split, with the team lead architect and senior engineers located at the main site of the organization ... This required significant communication taking place over geographical boundaries resulting in very inefficient development processes.

Together these quotes were condensed into one checklist question for AKM: *Is architectural design dependent on a single person / or centralized team?*

In our example answers, based on our own background and experience, we have considered the issues for a GSD company while using Scrum. We present the checklists and provided sample answers in interview format. Thus, please note that example answers are **not** intended as guidelines! The purpose of giving these example answers is to help practitioners achieve better focus on the given questions, as at first sight some questions may seem rather broad.

We found that there was a gap in current literature about where Conway's law applies - whether it goes beyond composition and structure. Furthermore, we raised the issue as to whether resources and skills could be taken into account when matching architecture to organization. While some answers are incomplete in terms of offering solutions, the checklist for organizational aspects in Table I should help the practitioner to fill the gap.

Cross-cutting themes from project management aspects and architecting are collected in Ways of Working, and the checklist for related concerns is given in Table II.

AKM issues have been the subject of intensive research, and they are an essential part of architectural design in GSD. Table III presents a checklist for AKM concerns. The challenges brought by global distance are particularly apparent in the AKM concerns, which highlight the need to ensure dispersion of knowledge and effective communication despite working in different time zones and countries.

There appeared to be an especially large gap in current literature regarding quality management, particularly when considering how critical quality is. Our SLR did not reveal any quality related practices. However, the checklist for quality management concerns as given in Table IV

will aid raising quality issues within the organization and identifying gaps in processes.

Change management is closely linked to quality management, as quality often deteriorates due to lack of compliance to (changing) requirements. A checklist for change management concerns is given in Table V.

Based on related work, modularity was a key concept in architecting in GSD. What remained unanswered was the level at which modularity is actually considered. The checklists for design practices in Table VI and for modularity related concerns in Table VII will support the practitioner in recognizing both the current working practices and potential gaps in the processes that the organization may want to consider.

5.2. Limitations and Validity

There are some limitations that must be addressed regarding our SLR, namely completeness, potential bias and limits to data synthesis. Limitations are discussed in subsection 5.2.1. In addition to limitations, as with any study, we must consider the potential threats to the validity of our results. Threats to validity, following Maxwell's categorization [67] and guidelines for fitting the categories to software engineering by Petersen and Gencel [68], are discussed in subsection 5.2.2.

5.2.1. Limitations

We have searched 6 of the most common and well-known databases containing software engineering literature. We have used as wide a range of search terms as possible, derived from the taxonomy by Smite et al. [62] to cover variations of terminology. Additionally, search terms were amended to suit particular databases after first searches if results were clearly skewed. All results or relevant results of which the authors were previously aware showed up in the search. Thus we have done our best to strive for an extensive coverage of the field. Using validation throughout the review process also ensured that all authors followed inclusion and exclusion criteria similarly. However, we do recognize that as the volume of material in this field is large, some work might still be left out unintentionally.

We also consider potential bias. The selection has been validated by two researchers (in addition to the researcher doing the initial selection) to ensure objectivity. Furthermore, the primary author of this work has not co-authored any of the included studies in the SLR, thus a researcher independent of the original work has reviewed every study. To investigate the potential bias of the primary studies we performed sensitivity analysis. The results of the analysis show that a majority of the studies originate from USA and Europe, and thus mainly reflect the Western viewpoint of GSD. Furthermore, we found a publishing peak, as a large number of studies were published in 2010. We did not find a particular reason for such a peak - it may just be that interest in GSD and particularly architecting aspects increased in the previous years and a large number of researchers were simultaneously active at that point.

Finally, there are limits regarding our data synthesis. We chose to use thematic analysis, which potentially lacks in transparency. To address these shortcomings, every step of the analysis has been carefully logged so choices for themes can be backtracked. Furthermore, each theme, practice, challenge and concern has been reviewed by at least two independent researchers. Therefore, how an individual article contributes to each theme has been validated. However, challenges and recommended practices may have lost clarity when condensing themes, and thematic borders may be unclear in parts, as there is significant overlap between certain practices and challenges particularly regarding organization, design practices and task allocation.

5.2.2. Threats to Validity

Descriptive and Theoretical Validity

Descriptive validity concerns accurate recording and presentation of the data on which conclusions are made. As the research method was a literature review and all material was thus in written form, descriptive validity threats are not a real concern in our case. Theoretical validity, in turn, concerns selection of subjects, instrumentation for data collection, definition of constructs, and other matters related to whether the data, and more importantly, the conclusions drawn from the data, can be trusted. Our data collection methodology was based on commonly agreed standards as presented by Kitchenham [59]. Definition of constructs and selection of subjects (defining population and inclusion and exclusion criteria) was done in an attempt to gain as wide a selection of studies as possible that would still answer the given research question as accurately as possible. A possible threat to validity relates to not considering the validity of the studies involved, i.e., we did not perform an evaluation of the validation or evaluation of the results in individual studies.

Interpretive Validity and Generalizability

Interpretive validity threats concern correct interpretations of the material. Validity threats related to interpretations have been dealt through rigorous validation of the thematic analysis by several objective researchers. First, the primary researcher who made the first interpretation of the material ("Does the literature discuss challenges or recommended practices?") had no previous publications in the area, and was thus an objective researcher. Second, other researchers checked verbatim quotes from studies and confirmed or disagreed with the interpretation of their content. Finally, the conceptual model, mapping of practices to the model and mapping of the low-level concerns to the high-level model were similarly validated. Thus, we are confident that validity threats to interpretive validity have been taken care of. Regarding generalizability, we are confident that internal generalizability (within the field of software engineering) is satisfied due to the wide

selection criteria for the studies. That is, there is no reason to believe that the results would not in general apply to companies in the field of software engineering involved in GSD. However, we do recognize that architectural design practices vary depending on company type, size and used processes, and unfortunately we did not have enough data to make conclusions on the link between this backgroud and used practices. External generalizability (to other fields) is not expected.

6. Conclusion

In this paper we conducted a systematic literature review of 55 papers that addressed our research questions which focused on challenges and recommended practices for architectural design in GSD.

We found that while there were no larger studies around the topic and individual papers only provided small contributions, through thematic analysis and collecting the information from the material we could identify several challenges and practices associated with our key themes: Organization (with sub-themes Structure and Resources), Ways of Working (with sub-themes AKM, Change Management, Quality Management), Design Practices, Modularity and Task Allocation.

In order to provide guidelines for GSD architecture design decisions, we mapped the challenges to practices found in the literature according to whether they were focused on project management or design decisions.

The core findings of the challenges and recommended practices were issues due to lack of communication and awareness, the difficulties and practices associated with task allocation, and the importance of following commonly accepted and proven architecture design practices. We also found that a clear majority of the identified challenges are the result of the distributed nature of the architecture design process and that the teams implementing this process and building the product are distributed. To address the challenges raised by GSD, we had expected to uncover design practices that are especially tailored for this context. However, only four of the nine practices we found are driven by global distribution:

1. Ensure that organization/work allocation is compliant with architecture design (P1)
2. Ensure knowledge of architectural artefacts and practices across sites with communication (P4)
3. Use a team of architects to collect and distribute the knowledge (P6)
4. Consider available resources from different sites in the design (P9)

Five out of nine of all the identified practices are quite common architectural guidelines that also apply when involved in collocated design. This raises the question: *Does architecting (design decisions) in GSD actually differ significantly from collocated architecting, or is it just that there is not enough information available about how to efficiently take the distributed nature of development work into account when doing the design?*

In addition to mapping the challenges and practices found in the literature, we elicited a concern framework around those challenges and practices to make them more accessible to practitioners. We further complemented the framework with checklists, which should aid architecting in practice. In our future work, we will report how our concern framework corresponds to current architecting practices in GSD, as we are now validating the findings of this SLR with an interview study of architects from seven different companies practicing GSD.

Acknowledgments

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).

Selected 55 Papers for the SLR

[SLR1] 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.

[SLR2] A. Mockus, D. M. Weiss, Globalization by chunking: a quantitative approach, IEEE Software 18 (2) (2001) 30–37.

[SLR3] G. Mustapic, A. Wall, C. Norstrom, I. Crnkovic, K. Sandstrom, J. Froberg, J. Andersson, Real world influences on software architecture - interviews with industrial system experts, in: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04), 2004, pp. 101–111.

[SLR4] 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.

[SLR5] D. Balasubramaniam, R. Morrison, R. M. Greenwood, B. Warboys, Flexible software development: From software architecture to process, in: Proceedings of The Working IEEE/IFIP Conference on Software Architecture, WICSA'07, 2007, pp. 14–14.

[SLR6] W. H. Huen, Systems engineering of complex software systems, in: Proceedings of the 37th Annual Frontiers in Education Conference, Global Engineering : Knowledge without Borders - Opportunities without Passports, Vols 1- 4, 2007, pp. 553–558.

[SLR7] D. A. Tamburri, P. Lago, C. Dorn, R. Hilliard, Architecting in networked organizations, in: Proceedings of the 2014 IEEE/IFIP Conference on Software Architecture (WICSA), 2014, pp. 247–250.

[SLR8] R. Britto, D. Šmite, L.-O. Damm, Software architects in large-scale distributed projects: An ericsson case study, IEEE Software, Special issue on Software Architect's Role in the Digital Age 33 (6) (2016) 48–55.

[SLR9] 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.

[SLR10] 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.

[SLR11] P. Lago, R. Farenhorst, P. Avgeriou, R. de Boer, V. Clerc, A. Jansen, H. van Vliet, The griffin collaborative virtual community for architectural knowledge management, in: Collaborative Software Engineering, Springer Berlin Heidelberg, 2010, pp. 195–217.

[SLR12] B. Tekinerdogan, S. Cetin, M. A. Babar, P. Lago, J. Mäkiö, Architecting in global software engineering, SIGSOFT Softw.Eng.Notes 37 (1) (2012) 1–7.

[SLR13] 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: Proceedings of the 2Nd International Workshop on Recommendation Systems for Software Engineering, ACM, 2010, pp. 36–40.

[SLR14] R. S. Sangwan, J. Ros, Architecture leadership and management in globally distributed software development, in: Proceedings of the First International Workshop on Leadership and Management in Software Architecture, ACM, New York, NY, USA, 2008, pp. 17–22.

[SLR15] 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.

[SLR16] J. D. Herbsleb, D. J. Paulish, M. Bass, Global software development at siemens: Experience from nine projects, in: Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), 2005, pp. 524–533.

[SLR17] J. Bosch, P. Bosch-Sijtsema, Coordination Between Global Agile Teams: From Process to Architecture, Springer Berlin Heidelberg, 2010, pp. 217–233.

[SLR18] F. Salger, G. Engels, A. Hofmann, Assessments in global software development: a tailorable framework for industrial projects, in: Proceedings of the 2010 ACM/IEEE 32nd International Conference on Software Engineering, 2010, pp. 29–38.

[SLR19] A. Lamersdorf, J. Munch, D. Rombach, A survey on the state of the practice in distributed software development: Criteria for task allocation, in: Proceedings of Fourth IEEE International Conference on Global Software Engineering (ICGSE'09), 2009, pp. 41–50.

[SLR20] J. A. Laredo, R. Ranjan, Continuous improvement through iterative development in a multi-geography environment, in: Proceedings of Third IEEE International Conference on Global Software Engineering (ICGSE'08), 2008, pp. 232–236.

[SLR21] J. Sauer, Architecture-Centric Development in Globally Distributed Projects, Springer Berlin Heidelberg, 2010, pp. 321–329.

[SLR22] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, M. Mazzara, From monolithic to microservices: An experience report from the banking domain, IEEE Software 35 (3) (2018) 50–55. doi:10.1109/MS.2018.2141026.

[SLR23] S. Beecham, J. Noll, I. Richardson, N. Ali, Crafting a global teaming model for architectural knowledge, in: Proceedings of the 5th IEEE International Conference on Global Software Engineering (ICGSE), 2010, pp. 55–63.

[SLR24] V. Clerc, P. Lago, H. van Vliet, Global software development: Are architectural rules the answer?, in: Proceedings of Second IEEE International Conference on Global Software Engineering (ICGSE'07), 2007, pp. 225–234.

[SLR25] S. Vaikar, M. M. Jha, F. Brunner, Using architectural constraints to drive software component reuse while adding and enhancing features, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE'16), 2016, pp. 139–143.

[SLR26] 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.

[SLR27] T. Burity, G. Elias, A quantitative, evidence-based approach for recommending software modules, in: Proceedings of ACM Symposium of Applied Computing (SAC'15), 2015, pp. 1449–1456.

[SLR28] R. P. dos Santos, C. M. L. Werner, Reuseecos: An approach to support global software development through software ecosystems, in: IEEE Seventh International Conference on Global Software Engineering Workshops (ICGSEW'12), 2012, pp. 60–65.

[SLR29] J. Bang, D. Popescu, G. Edwards, N. Medvidovic, N. Kulkarni, G. M. Rama, S. Padmanabhuni, Codesign: A highly extensible collaborative software modeling framework, in: Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2, ACM, New York, NY, USA, 2010, pp. 243–246.

[SLR30] M. Cataldo, C. Shelton, C. Yongjoon, H. Yun-Yin, V. Ramesh, D. Saini, W. L.-Y. Wang, Camel: A tool for collaborative distributed software design, in: Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE'09), 2009, pp. 83–92.

[SLR31] A. Corry, K. Hansen, D. Svensson, Traveling architects - a new way of herding cats, in: Proceedings of the International Conference on the Quality of Software Architectures (QoSA 2006), Springer Berlin Heidelberg, 2006, pp. 111–126.

[SLR32] B. M. Yildiz, B. Tekinerdogan, Architectural viewpoints for global software development, in: Proceedings of the 2011 Sixth IEEE International Conference on Global Software Engineering Workshop (ICGSEW), 2011, pp. 9–16.

[SLR33] B. M. Yildiz, B. Tekinerdogan, S. Cetin, A tool framework for deriving the application architecture for global software development projects, in: Proceedings of the 2012 Seventh IEEE International Conference on Global Software Engineering (ICGSE), 2012, pp. 94–103.

[SLR34] O. Tufekci, S. Cetin, A. Arifoglu, Proposing a federated approach to global software development, in: Proceedings of the Fourth International Conference on Digital Society, 2010. ICDS'10., 2010, pp. 150–157.

[SLR35] 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'16), 2016, pp. 109–118.

[SLR36] K. Popovič, Z. Hocenski, G. Martinovic, Do the software architects get the needed support for the job they perform?, in: Proceedings of the 2010 Fifth International Conference on Software Engineering Advances (ICSEA), 2010, pp. 123–128.

[SLR37] M. Paasivaara, C. Lassenius, Scaling scrum in a large globally distributed organization: a case study, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE'16), 2016, pp. 75–83.

[SLR38] V. Clerc, P. Lago, H. van Vliet, Architectural knowledge management practices in agile global software development, in: Proceedings of the Fourth IEEE International Conference on Global Software Engineering Workshop (ICGSEW'11), 2011, pp. 1–8.

[SLR39] G. Borrego, A. Morán, R. Palacio, O.M.Rodriguez, Findings on agsd architectural knowledge sharing, in: Proceedings of the 11th IEEE International Conference on Global Software Engineering (ICGSE'16), 2016, pp. 193–194.

[SLR40] D. Rost, M. Naab, C. Lima, C. von Flach Garcia Chavez, Software architecture documentation for developers: A survey, in: Proceedings of the European Conference on Software Architecture (ECSA'13), Vol. 7957, 2013, pp. 72–88.

[SLR41] G. Borrego, A. L. Morn, R. Palacio, Preliminary evaluation of a tag-based knowledge condensation tool in agile and distributed teams, in: 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE), 2017, pp. 51–55. doi:10.1109/ICGSE.2017.14.

[SLR42] 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.

[SLR43] V. Clerc, E. de Vries P. Lago, Using wikis to support architectural knowledge management in global software development, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, ACM, 2010, pp. 37–43.

[SLR44] J. M. Bass, Artefacts and agile method tailoring in large-scale offshore software development programs, Information and Software Technology 75 (2016) 1–16.

[SLR45] J.M.Bass, How product owner teams scale agile methods to large distributed enterprises, Empirical Software Engineering 20 (2015) 1525–1557.

[SLR46] H. R. de Faria, G. Adler, Architecture-centric global software processes, in: Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE'06), 2006, pp. 241–242.

[SLR47] L. G. Bratthall, R. van der Geest, H. Hofmann, E. Jellum, Z. Korendo, R. Martinez, M. Orkisz, C. Zeidler, J. S. Andersson, Integrating hundred's of products through one architecture – the industrial it architecture, in: Proceedings of the 24th International Conference on Software Engineering, IEEE, 2002, pp. 604–614.

[SLR48] J. Rudzki, I. Hammouda, T. Mikkonen, Ensuring architecture conventions in multi-site development, in: Proceedings of 32nd Annual IEEE International Computer Software and Applications Conference, 2008 (COMPSAC '08), 2008, pp. 339–346.

[SLR49] G. Caprihan, Managing software performance in the globally distributed software development paradigm, in: Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE), 2006, pp. 84–91.

[SLR50] J. Herbsleb, R. E. Grinter, Architectures, coordination, and distance: Conway's law and beyond, IEEE Software 16 (5) (1999) 63–70.

[SLR51] V. Clerc, Do architectural knowledge product measures make a difference in gsd?, in: Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE'09), 2009, pp. 382–387.

[SLR52] J. M. Bass, Agile method tailoring in distributed enterprises: Product owner teams, in: Proceedings of the IEEE 8th International Conference on Global Software Engineering (ICGSE 2013), 2013, pp. 154–163.

[SLR53] N. Ali, S. Beecham, I. Mistrik, Architectural knowledge management in global software development: A review, in: Proceedings of ICGSE'10, IEEE, 2010, pp. 347–352.

[SLR54] M. Cataldo, J. D. Herbsleb, K. M. Carley, Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity, in: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, 2008, pp. 2–11.

[SLR55] 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.

References

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

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

[3] V. Clerc, P. Lago, and H. van Vliet, "Global software development: Are architectural rules the answer?" in *Proceedings of Second IEEE International Conference on Global Software Engineering (ICGSE'07)*, 2007, pp. 225–234.

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

[5] A. M. D. Santana, F. Q. B. da Silva, R. C. G. de Miranda, A. A. Mascaro, T. B. Gouveia, C. v. F: Monteiro, and 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.

[6] M. Bano, D. Zowghi, and N. Sarkissian, "Empirical study of communication structures and barriers in geographically distributed teams." *IET Software*, vol. 10, no. 5, pp. 147–153, 2016.

[7] S. Imtiaz and N. Ikram, "Dynamics of task allocation in global software development," *J. Softw. Evol. and Proc*, 2016.

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

[9] C. Wohlin, D. Šmite, and N. B. Moe, "A general theory of software engineering: Balancing human, social and organizational capitals," *Journal of Systems and Software*, vol. 109, pp. 229–242, 2015.

[10] J. Bosch, *Design & Use of Software Architectures - Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

[11] D. L. Parnas, "A technique for software module specification with examples," *Commun. ACM*, vol. 15, no. 5, pp. 330–336, 1972.

[12] ——, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[13] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The modular structure of complex systems," *IEEE Trans. Software Eng*, vol. 11, no. 3, pp. 259–266, 1985.

[14] G. Mustapic, A. Wall, C. Norstrom, I. Crnkovic, K. Sandstrom, J. Froberg, and J. Andersson, "Real world influences on software architecture - interviews with industrial system experts," in *Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA'04)*, 2004, pp. 101–111.

[15] I. Kwan, M. Cataldo, and D. Damian, "Conway's law revisited: The evidence for a task-based perspective," *IEEE Software*, vol. 29, no. 1, pp. 90–93, 2012.

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

[17] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: A systematic review," *Empirical Software Engineering*, vol. 15, no. 1, pp. 91–118, 2010.

[18] E. Ó Conchúir, P. øAgerfalk, H. Holmstrom, and B. Fitzgerald, "Global software development: Never mind the problems - where are the benefits?" *Comm. of the ACM*, vol. 52, pp. 127–131, 2009.

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

[20] J. Kroll, I. Richardson, J. L. N. Audy, and J. Fernandez, "Handoffs management in follow-the-sun software projects: A case study," in *Proceegings of the 2014 47th Hawaii International Conference on System Science (HICSS'14), Waikoloa, HI*, 2014, pp. 331–339.

[21] J. Kroll, I. Richardson, and J. L. N. Audy, "FTS-SPM: A software process model for follow the sun development: Preliminary results," in *Proceedings of the 2014 IEEE International Conference on Global Software Engineeering Workshops*, 2014, pp. 21–26.

[22] J. Kroll, J. Audy, and R. Prikladnicki, "Mapping the evolution of research on global software engineering: a systematic litera-

ture review," in *Proceedings of the International Conference on Enterprise Information Systems (ICEIS?11)*, 2011.

[23] H. Holmström, E. Ó Conchúir, P. Ågerfalk, and B. Fitzgerald, "Global software development challenges: A case study on temporal, geographical and socio-cultural distance," in *Proceedings of International Conference on Global Software Engineering (ICGSE '06)*, 2006, pp. 3 –11.

[24] P. J. Ågerfalk, B. Fitzgerald, H. Holmström, B. Lings, B. Lundell, and E.Ó Conchúir, "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.

[25] C. Casey and I. Richardson, "Implementation of global software development: A structured approach," *Journal of Software Evolution and Process*, vol. 14, no. 5, pp. 247–262, 2009.

[26] J. Noll, S. Beecham, and I. Richardson, "Global software development and collaboration: barriers and solutions," *ACM Inroads*, vol. 1, no. 3, pp. 66–78, 2011.

[27] R. Prikladnicki and J. L. N. Audy, "Process models in the practice of distributed software development: A systematic review of the literature," *Information and Software Technology*, vol. 52, no. 8, pp. 779–791, 2010.

[28] A. B. Marques, R. Rodrigues, and T. Conte, "Systematic literature reviews in distributed software development: A tertiary study," in *Proceedings of the 2012 IEEE Seventh International Conference on Global Software Engineering (ICGSE)*, 2012, pp. 134–143.

[29] F. Lanubile, D. Damian, and H. D. Oppenheimer, "Global software development: technical, organizational and social challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 28, no. 6, p. 2, 2003.

[30] M. Babar and C. Lescher, "Global software engineering: Identifying challenges is important and providing solutions is even better," *Information and Software Technology*, vol. 56, no. 1, pp. 1–5, 2014.

[31] M. Jimenez, M. Piattini, and A. Vizaino, "Challenges and improvements in distributed software development: a systematic review," *Advances in Software Engineering*, pp. 1–16, 2009.

[32] 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.

[33] R. Britto, D. Šmite, and L.-O. Damm, "Software architects in large-scale distributed projects: An Ericsson case study," *IEEE Software, Special issue on Software Architect's Role in the Digital Age*, vol. 33, no. 6, pp. 48–55, 2016.

[34] ISO/IEC/IEEE, *Systems and software engineering - Architecture description*, Std., Rev. 42010:2011, 2011.

[35] D. Garlan, "Software architecture: a roadmap," in *Proceedings of the Conference on The Future of Software Engineering (ICSE'00)*. New York, NY, USA: ACM, 2000, pp. 91–101.

[36] I. Crnkovic, "Component-based software engineering - new challenges in software development," *Software Focus*, vol. 2, no. 4, pp. 127–133, 2001.

[37] J. Herbsleb and R. E. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *IEEE Software*, vol. 16, no. 5, pp. 63–70, 1999.

[38] A. Avritzer, D. Paulish, Y. Cai, and K. Sethi, "Coordination implications of software architecture in a global software development project," *J. Syst. Software*, vol. 83, no. 10, pp. 1881–1895, 2010.

[39] M. Shaw and D. Garlan, *Software Architecture - Perspectives on an Emerging Discipline*. Upper Saddle River, NJ: Prentice Hall, 1996.

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

[41] "Rest," http://www.restapitutorial.com/lessons/whatisrest.html, accessed: 2017-02-28.

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

[43] E. Wolff, *Microservices: Flexible Software Architecture*. Addison-Wesley, 2016.

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

[45] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," *Cloud computing*, pp. 626–631, 2009.

[46] G. Kulkarni, "Cloud computing-software as service," *International Journal of Cloud Computing and Services Science*, vol. 1, no. 1, p. 11, 2012.

[47] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[48] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1995.

[49] G. Hohpe and B. Woolf, *Enterprise Integration Patterns*. Addison-Wesley, 2004.

[50] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects. Vol. 2*. John Wiley & Sons, 2013.

[51] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley, 1996.

[52] A. G. J. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA 2005)*, 2005, pp. 109–119.

[53] U. van Heesch, V.-P. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison, "Decision-centric architecture reviews," *IEEE Software*, vol. 31, no. 1, pp. 69–76, 2014.

[54] L. Lundberg, J. Bosch, D. Häggander, and P. O. Bengtsson, "Quality attributes in software architecture design," in *Proceedings of the Third International Conference on Software Engineering and Applications*, 1999, pp. 353–362.

[55] S. Vaikar, M. M. Jha, and F. Brunner, "Using architectural constraints to drive software component reuse while adding and enhancing features," in *Proceedings of ICGSE'16*. IEEE, 2016, pp. 139–143.

[56] N. Ali, S. Beecham, and I. Mistrik, "'architectural knowledge management in global software development: A review," in *Proceedings of ICGSE'10*. IEEE, 2010, pp. 347–352.

[57] S. S. M. Fauzi, P. L. Bannerman, and 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.

[58] A. Mishra and 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.

[59] B. Kitchenham, "Procedures for performing systematic reviews," p. 28 pages, 2004.

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

[61] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Protocol of a systematic literature review of motivation in software engineering," UK, 2006.

[62] D. Šmite, C. Wohlin, Z. Galvina, and R. Prikladnicki, "An empirically based terminology and taxonomy for global software engineering," *Empirical Software Engineering*, vol. 19, no. 1, 2014.

[63] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[64] D. Cruzes and T. Dybå, "Research synthesis in software engineering: a tertiary study," *Information and Software Technol-*

ogy, vol. 53, pp. 440–455, 2011.

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

[66] K. Charmaz, *Constructing grounded theory.* Sage, London, 2006.

[67] J. A. Maxwell, "Understanding and validity in qualitative research," *Harvard educational review*, vol. 62, pp. 279–301, 1992.

[68] K. Petersen and C. Gencel, "Worldviews, research methods, and their relationship to validity in empirical software engineering research," in *Proceedings of the 2013 Joint Conference of the 23Nd International Workshop on Software Measurement (IWSM) and the 8th International Conference on Software Process and Product Measurement*, ser. IWSM-MENSURA'13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 81–89. [Online]. Available: http://dx.doi.org/10.1109/IWSM-Mensura.2013.22

Appendix A. Review protocol, search strings and validation

The steps for our protocol (after defining research questions) were as follows:

1. Define population, intervention, outcomes of relevance and experimental designs of interest;
2. Derive major terms from the questions by identifying the population, intervention and outcome;
3. Identify alternative spellings and synonyms for major terms;
4. Check the keywords in any relevant papers we already have;
5. When databases allow, use the Boolean OR to incorporate alternative spellings and synonyms;
6. When databases allow, use the Boolean AND to link the major terms from population, intervention and outcome.

Performing step 1 provided us with the following definitions: Population: Practitioners involved in either using the architecture, or designing the architecture in a GSD environment. Intervention: Architectural design for GSD, Architectural decisions in GSD, Architectural knowledge management in GSD. Outcomes of relevance: How software architecture affects task allocation, How software architecture affects product quality, How organizational structure affects software architecture design, How task allocation affects product quality. Experimental designs of interest: Empirical studies, experience reports, expert observations, reviews.

Performing step 2 provided us with the basic search terms: RQ1: software architecture, design, challenges, globally distributed sites, developing software. RQ2: software architecture, design, recommended practices, geographically distributed sites, software development.

Additionally, we were particularly interested if we could find differences between between the architectural design practices used in collocated and those used in globally distributed sites. Thus we also added the search term "collocated". An additional search was performed with a search string combining terms for RQ2 and "collocated".

For step 3 we identified alternative spellings and terms. For example, we decided that process and practice would be combined as synonyms for the same term. After also checking keywords (step 4), and using Boolean operators (step 5) we got the following terms to use in our search:
Software architecture: software architect* OR system architect* OR software product architect*
Design: design* OR develop* OR produc*
Software development: (software OR system*) AND (develop* OR engineer* OR produc*)
Collocated: collocate* OR same site* OR samesite OR same-site* OR one site* OR one-site* OR onesite* OR local* OR same location
Geographically distributed sites: GSD OR DSD OR GSE OR DSE OR (distribut* OR multisite* OR global OR multi-site* OR offsite OR off-site OR offshore* OR off-shor* OR insource* OR in-sourc* OR multisource* OR multi-sourc* OR farshor* OR far-shor* OR many site* OR different location* OR distributed sites)
Recommended: good OR best OR recommend* OR prefer* OR common Process or Practice: practice* OR guideline* OR framework OR model OR pattern OR custom OR way OR process OR procedure OR protocol*
Challenges: challeng* OR problem* OR restriction* OR constrain* OR difficult* OR question*
The actual search strings were constructed by combining all given spellings and synonyms of one term by using the OR operator, and combining all terms related to a research question with the AND operator (step 6). For example, the search string for RQ1 was thus:

(software architect* OR system architect* OR software product architect*) AND (design* OR develop* OR produc*) AND (challeng* OR problem* OR restriction* OR constrain* OR difficult* OR question*) AND (GSD OR DSD OR GSE OR DSE OR (distribut* OR multisite* OR global OR multi-site* OR offsite OR off-site OR offshore* OR off-shor* OR insource* OR in-sourc* OR multisource* OR multi-sourc* OR farshor* OR far-shor* OR many site* OR different location* OR distributed sites) AND software AND (engineer* OR develop*) AND ((software OR system*) AND (develop* OR engineer* OR produc*))

We searched the following databases (in this order):

- IEEE Explore
- ACM Digital library
- SpringerLink
- ScienceDirect
- Wiley digital library
- ISI Web of Knowledge

Google Scholar was excluded as it would not allow a detailed enough search string. For some databases we could not use all the search terms due to limitations in the number of allowed terms, and also could not combine different searches with subsets of search terms. In those cases we would use search terms related to software architecture, design, collocated and distributed sites, and use the following inclusion criteria:

Include only papers with good/best/recommended practices/processes/framework/model/pattern or challenges

Review process validation

The searches produced a total of 1197 unique references. One author was primarily in charge of selecting the papers, but as per the four eyes principle, selections were validated by other authors. The selection process and validation proceeded as follows. The primary author went through each paper's title and abstract, and made a decision to either

include or exclude it based on the given criteria. After this, two other authors were both given 100 paper's titles and abstracts each, and they performed a similar selection process. The papers given for validation were selected at random, only making sure that there were a representative number of papers the primary author had accepted to ensure comparison of decisions. The decisions of different authors were then checked we set a limit that initially we should agree on 80% of the cases; otherwise the criteria and/or the selection process should be re-investigated. If two authors had made a different decision (one said to include the paper, the other to exclude), the authors would negotiate and justify their choices. If they could not reach a consensus on what to do, the third author would have the final say. After the validation phase, 135 papers were included for the next phase. Next, the full texts of all included papers would be read. A paper would then be included if it could provide direct answers to the research questions and also still fit the inclusion/exclusion criteria. Once again the selection was validated by giving a set of full papers to two other authors each to read. These papers were again selected randomly, only ensuring that there were ones that the primary author had accepted as well as rejected ones. Conflicting selections were negotiated similarly as in the previous phase. After going through the full texts, we had 46 references to include in the review. To cover all potential studies, we also checked the reference sections within the selected 46 papers i.e., performed "backward snowballing, as recommended by Kitchenham [59]. From the reference lists we found 33 potential titles that had not been included in our original search. Note, that for review articles some papers cited in the review articles were selected for the SLR and some not, thus the review articles contain more information than the articles cited by them that are included as individual articles in the SLR. A full-text screening was performed for all these title which resulted in an additional 9 studies for the review. Thus, the final number of the reviewed papers published by May 2018 is 55.

Appendix B: Checklists

Table I: Checklist for organizational concerns

Question	**Rationale**	**Example answer in a company using Scrum**
What kind of organizational aspects are considered during design - are all necessary aspects covered?	Design is more likely to stay stable during development and task allocation is easier if organizational aspects are considered beforehand.	*We have considered organization reporting hierarchy, business goals, practices for component reuse and application of Scrum. Notice that have not considered how to handle maintenance and changes with a distributed team of architects.*
Does the software structure guide the organization or vice versa? Do the structures match?	Conway's law suggests that the architecture will mirror the organization (communication structure). More specifically task allocation will likely mirror the communication structure.	*We have considered where teams are allocated, which teams will handle which tasks and which teams will need to communicate with each other when designing – thus taken care to separate modules or tasks accordingly and that the structures match.*
What kind of practices are there to scale down tasks - do they work?	Poorly defined tasks may be too large for one team (or person) to handle, resulting in scheduling and synchronization problems and possibly lead to tasks across sites when extra resources are required.	*We allocate large tasks per site and the site manager takes care of scaling down the tasks per team.*
On what scale is work structure considered?	Relates to previous question on scaling down work items.	*See above.*
Are the available resources considered/do they act as a driving force for the design?	Recognizing resources helps assure that product architecture matches the design. Resources here mean primarily skills, but also available personnel and schedule (effort), budget, available components for reuse, licenses, etc.	*Architecture is designed with available personnel, reusable components and platform in mind.*
How is the distribution of the resources considered?	Here resources means people and skills. Distribution of skills is important to recognize, if some tasks require several specialized skills that span across sites. Sharing skills is more difficult between sites.	*Distribution of skills is not considered beforehand. Will need to possibly rethink work structure.*
Are there sufficient practices to match code for available resources?	There is a need to recognize skills and available manpower and scale down work items to a sufficiently fine-grained level when planning task allocation. Tasks should be doable by one (single-site) team.	*We allocate large tasks per site and the site manager takes care of scaling down the tasks per team. Site manager knows his team members and their skills.*
Are there working practices to align decisions to organization structure?	Large-scale design decisions may affect required skills and therefore also task allocation as much as modular structure.	*Architecture is evaluated after each design iteration to check that decisions comply with organization.*
Are there working practices to handle misaligned interests?	Undesirability of some tasks is a problem especially in remote sites when rationale behind some tasks may not be understood or teams feel their expertise does not correspond with the task's requirements.	*Site managers will assign tasks. The architect is responsible that all tasks are completed in schedule. If necessary, the architect will either aid the task manager in breaking down the task to the teams, or seek for other teams (if possible).*
How are potential conflicts (e.g., in schedule) identified and handled?	Conflicts are more difficult to spot when teams are working in remote sites and thus if one team is experiencing problems with, e.g., keeping to agreed schedule, other sites may not become aware in time.	*Tasks are scheduled in weekly Scrum meetings between architects and site managers. Site managers will keep track of their teams' status and report if schedule does not keep.*

Table II: Checklist for concerns related to ways of working

Question	**Rationale**	**Example answer in a company using Scrum**
Are all stakeholders identified and the appropriate architecture view composed for their purposes?	It is important to identify all the stakeholders who require architectural information. Stakeholders may have different backgrounds, and thus different views may need to be considered for understandability and promoting awareness.	*Architecture is communicated only to developers on remote sites. Other stakeholders are met on main site and appropriate views used.*
Are diagrams drawn with the stakeholder in mind?	Follow-up to the previous question - considering the stakeholders background and agenda and choosing the diagram accordingly will help promote awareness and reach a better level of understanding.	*Less technical views are used for business purposes. Developers are assumed to be aware of common technical notations.*
What kind of diagrams are used?	Many architects tend to favor UML, which are not, however, understood by less technical people or even developers with significantly different backgrounds in terms of education.	*Freeform and UML.*
Are you using methods to calculate coupling? Should such methods be used?	Coupling is an easy measurement to calculate dependencies between components. Tools and analysis methods can be used to calculate coupling values if dependencies within the architecture are otherwise overlooked or difficult to find.	*Coupling is calculated automatically from code.*
Are there visualizations of decisions or metrics?	Calculating quality metrics from the architecture may help finding potential bottlenecks both in terms of quality and task allocation. Visualizations can aid grasping large-scale decisions.	*A decision dependency diagram is created based on links in artefact repository and wiki.*
Is collaborative modeling an option?	Collaborative modeling can help in situations where teams tend to incorrectly assume functionality of components developed by others.	*We have chosen not to use collaborative modeling.*

Table III: Checklist for AKM concerns

Question	**Rationale**	**Example answer in a company using Scrum**
Are all teams and their relevant members aware of the architectural design and decisions?	Not all developers need to be aware of the complete architecture design, and it is often sufficient that the teams know their own modules and interfacing modules. However, there should be at least someone in the team who is aware of the architecture and the rationale behind the decisions. This is not self-evident in distributed development.	*Each site has representative architect, who is responsible for further disseminating architectural knowledge on that site.*
Who is responsible for the creation of architectural artefacts?	Evidence shows that particularly when decisions are delegated there is uncertainty about responsibilities.	*The team of architects (one for each site) is responsible for creation of architectural artefacts.*
Where are the artefacts stored?	Artefacts should be stored to one common repository for easy access and dissemination.	*There is a repository where diagrams, code and other such artefacts are stored. Additionally, a wiki is used for documenting.*
Do all the teams have access to architectural artefacts?	Access to artefacts promotes awareness and increases trust as teams do not feel left out or that one site is favored over others.	*Both the wiki and the repository are accessible by all project members.*
Are the artefacts understood by (representatives of) all teams?	Architects may present some decisions as they were directed to only other architects.	*All artefacts are reviewed on team level and thus they should be understandable. If a team member requires information about an artefact the on-site architect is responsible for clarifications.*
Is architectural documentation detailed enough?	Particularly remote team members need more specific documentation to compensate lack of face-to-face discussions.	*Lack of detail usually becomes apparent in daily Scrums, where team members ask for clarifications. We could have a better process of ensuring enough detail while preparing the document.*
Is architectural design dependent on a single person/ or centralized team?	Relying heavily on one chief architect may end up being a bottleneck, as the chief architect will be overwhelmed with information requests. A centralized architecture team helps in answering questions, but is still not as affective as a distributed team to promote awareness.	*In our case we use a distributed team of architects, one for each site.*
What are the current knowledge management practices?	Knowledge management practices should be reviewed and evaluated for sufficiency at the start of each project.	*A shared repository, a common wiki and a distributed team of architects who promote awareness at sites.*
How is the design process arranged with distributed architects?	While a distributed team of architects aids in terms of awareness, it brings more challenges to the actual design process.	*In the beginning of design the architects meet in a collocated design workshop. After that they have first online meetings twice a week, and then weekly. Architects will gather for a first review of the architecture, after which online meetings are held again more often. The need for communications fluctuates during the design depending on the state of the design.*
What are the architects' backgrounds and main drivers?	It is important to include a an architect with sufficient technical background knowledge who will be able to spot possible difficulties in development more easily.	*All architects have a technical background.*

Table IV: Checklist for quality management concerns

Question	**Rationale**	**Example answer in a company using Scrum**
How is quality ensured if design decisions are delegated to teams?	Small-level architectural decisions are commonly delegated to the teams in charge of developing the corresponding piece of software. Quality may deteriorate if these decisions are not checked in a coordinated fashion.	*We use automated unit tests to check functional correctness. Otherwise the on-site architect is in charge for checking team level decisions.*
Who is responsible for the quality concerns?	Quality assurance is a vital but often overlooked part of the design process. In a distributed setting where responsibilities easily become unclear, quality assurance is particularly something that needs a clearly assigned person.	*The architect team is responsible for meeting the quality requirements.*
How is quality measured? Are measures only for functional tests or are non-functional requirements addressed as well?	Often quality is only measured as how many tests are passed, and tests check for functional quality (i.e., meeting functional requirements). Measurements for meeting non-functional requirements should be considered as well.	*In addition to automated tests we have performance tests. Architecture evaluations are also held to check conformance to quality demands.*
How is data and state management organized?	Centralized data and state management may lead to unintentional dependencies, causing deteriorated quality and requiring more effort in development due to added communication needs.	*Data and state management is done considering the best choice for the product. Individual databases are used for testing purposes per site.*

Table V: Checklist for change management concerns

Question	**Rationale**	**Example answer in a company using Scrum**
Are possible changes in communication structure considered in the design structure?	Conway's law states that organizations tend to implement designs that mirror their (communication structure). This implies that critical changes in communication structure should be reflected in the architecture as well.	*We are using Scrum, and thus some architectural decisions are left open until later in the design. Within the limits of the core design we may thus address changes in communication structure to a certain level.*
Is the architecture understandable in a changing organization?	One should be aware that if key resources (member(s) of the architecture team) are no longer available, the architecture should still be understandable. If the organization undergoes significant structural changes, an architecture that was designed to reflect the previous organization structure may also seem complex in the new organization.	*Having a team of architects ensures a widespread knowledge of the architecture. Drastic changes in the organization structure will lead to a review of the architecture.*
Is the architecture compliant with the requirements?	Functional requirements have a tendency to change over time, and quality requirements are refined as well. Design should be checked for conformance regularly.	*Architects review changes in requirements in their weekly meetings and possible changes to design are disseminated to teams in Scrum meetings by on-site architects.*
Is ownership of architectural elements agreed upon?	In case decisions are largely delegated to teams, there may be confusion on who is responsible for particular architectural elements.	*The team of architects share ownership and decide amongst themselves a more detailed division of responsibilities.*
How is synchronization handled?	Synchronization relates to synchronizing tightly coupled work items and states as well as release synchronization. Not all are obvious.	*Tightly coupled items are managed by architects. Release synchronization is handled through Scrum practices.*
Is the business process known and considered while doing the design?	The business process may affect how development is done (in terms of using resources, handling requirements, etc.), and, e.g., how off-the-shelf components may be used.	*The team of architects communicates freely with the business management side of the organization and is aware of the limits it sets for design.*

Table VI: Checklist for design practice concerns

Question	**Rationale**	**Example answer in a company using Scrum**
What are the common design agreements that consider distributed development?	Architects often do design based on their previous experiences on projects and commonly agreed practices (found in, e.g., literature). It is good to inspect whether these practices take into account the distributed nature of development work.	*Extra effort is put to interface design and definitions to ensure that interfaces are implemented according to expectations and agreements. Architecture should be divided into small components but also kept simple.*
What is required from an interface to make it "well defined"?	"Well-defined" interfaces is a common solution for separation of components/modules/tasks, but what is needed for an interface to be well-defined varies.	*All operation must have inputs and outputs defined and a state diagram drawn.*
How much instability is caused due to the distributed nature of development?	Instability of architecture should be minimized, as it effects quality of both product and work, and may hinder efficient task allocation.	*Using a team of architects promotes stability as design and development is coordinated at each site.*
Who defines prioritization?	Knowing dependencies within the architecture is important for correct prioritization of design and development tasks.	*Prioritization is defined by architects and site managers, who may delegate small scale prioritization to team leaders.*
How are design principles recorded and their use enforced?	Each design decision should be based on rationale or principles. Particularly in a changing organization it is important to also log the rationale somewhere to ensure design quality and understandability. Furthermore, if design decisions are delegated to teams it should be ensured that they follow the same design principles as the architects.	*Architectural decisions are documented in a wiki, where also rationale is included. Artefacts in the common repository are linked to the wiki. Each architect is committed to the common principles.*
How are assumptions handled?	Lack of communication often leads to incorrect assumptions on what other teams are developing and how.	*Each architect is aware of the big picture and is responsible for giving enough information on-site.*

Table VII: Checklist for modularity concerns

Question	**Rationale**	**Example answer in a company using Scrum**
What kinds of mechanisms are used for enhancing low coupling?	Low coupling is essential for separation of tasks.	*We follow design principles that reach for as independent components as possible, if there are many dependencies, components are split to smaller entities if possible.*
How is it verified that all dependencies are known?	While usage dependencies between components may be rather easy to find, other dependencies exist that may be more difficult to find.	*Dependencies will become apparent at development stage the latest, and the information will be forwarded to the representative architect on site.*
How are interdependencies between decisions considered?	Choosing libraries, COTS components, etc. may affect further design choices.	*Dependencies between decisions are identified and noted in the Wiki documentation.*