System and software architecting harmonization practices in ultra-large-scale systems of systems

Cadavid Rengifo, Hector; Andrikopoulos, Vasilios; Avgeriou, Paris; Broekema, P. Chris

Published in:
Information and Software Technology

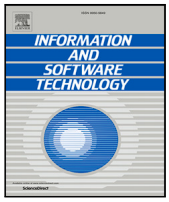
DOI:
10.1016/j.infsof.2022.106984

Information and Software Technology 150 (2022) 106984

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



System and software architecting harmonization practices in ultra-large-scale systems of systems: A confirmatory case study

Héctor Cadavid [a,c,*], Vasilios Andrikopoulos [a], Paris Avgeriou [a], P. Chris Broekema [b,d]

[a] Department of Computer Science, Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands
[b] Netherlands Institute for Radio Astronomy (ASTRON), Dwingeloo, The Netherlands
[c] Escuela Colombiana de Ingeniería, Bogotá, Colombia
[d] University of Cambridge, United Kingdom of Great Britain and Northern Ireland

ARTICLE INFO

Keywords:
Systems of systems
SoS architecting
Confirmatory case study
Empirical software engineering
Scientific instruments
Qualitative research

ABSTRACT

Context: The challenges posed by the architecting of System of Systems (SoS) has motivated a significant number of research efforts in the area. However, literature is lacking when it comes to the interplay between the disciplines involved in the architecting process, a key factor in addressing these challenges.
Objective: This paper aims to contribute to this line of research by confirming and extending previously characterized architecting harmonization practices from Systems and Software Engineering, adopted in an ultra-large-scale SoS.
Methods: We conducted a confirmatory case study on the Square-Kilometre Array (SKA) project to evaluate and extend the findings of our exploratory case on the LOFAR/LOFAR2.0 radio-telescope projects. In doing so, a pre-study was conducted to map the findings of the previous study with respect to the SKA context. A survey was then designed, through which the views of 46 SKA engineers were collected and analyzed.
Results: The study confirmed in various degrees the four practices identified in the exploratory case, and provided further insights about them: (1) the friction between disciplines caused by long-term system requirements, and how they can be ameliorated through intermediate, short-term requirements; (2) the way design choices with a cross-cutting impact on multiple agile teams have an indirect impact on the system architecture; (3) how these design choices are often caused by the criteria that guided early system decomposition; (4) the seemingly recurrent issue with the lack of details about the dynamic elements of the interfaces; and (5) the use of machine-readable interface specifications for aligning hardware/software development processes.
Conclusions: The findings of this study and its predecessor support the importance of a cross-disciplinary view in the Software Engineering research agenda in SoS as a whole, not to mention their value as a convergence point for research on SoS architecting from the Systems and Software Engineering standpoints.

1. Introduction

The concept of System of Systems (SoS) is extensively used in application domains like defense [1], manufacturing [2,3], emergency systems [4,5], energy [6,7], and health care [8,9] to describe *a collection of systems that cooperate to fulfill a goal or to provide new capabilities* [10]. By definition, most SoS, like those in the aforementioned domains, are characterized by different degrees of *Managerial and Operational Independence*, *Diversity*, *Heterogeneity*, *Emergence* and *Belonging* as proposed by Maier [11] and Boardman et al. [12]. Such characteristics make SoS a powerful concept, but at the same time pose significant challenges to their architecting process. For example, the dynamic nature of an SoS makes it difficult to anticipate its behavior at design time, and therefore quality requirements are difficult to address [13,14].

Although there have been significant research efforts to address these challenges [15], there is still little research on a key element of the SoS architecting process: the interplay between the disciplines involved in it, namely Systems Engineering and Software Engineering. To contribute towards addressing this research gap, we have conducted a series of studies which, together, aimed at: (1) identifying and characterizing the pain points of the software architecting processes

* Corresponding author at: Department of Computer Science, Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands.
E-mail addresses: h.f.cadavid.rengifo@rug.nl (H. Cadavid), v.andrikopoulos@rug.nl (V. Andrikopoulos), p.avgeriou@rug.nl (P. Avgeriou), broekema@astron.nl (P.C. Broekema).

https://doi.org/10.1016/j.infsof.2022.106984
Received 2 December 2021; Received in revised form 9 June 2022; Accepted 13 June 2022
Available online 22 June 2022

in large-scale SoS where development is usually driven by Systems Engineering (SE) [16]; and (2) identifying and characterizing the best practices to harmonize said architecting processes. The first study of the series [17] confirmed that major integration and operational problems in SoS are indeed linked to the way different disciplines work together on their architecting process. Particular instances of such problems – related to requirements and interfaces – were identified. In the second study [18], we identified a number of best practices that engineers and architects have adopted exactly to address the aforementioned interdisciplinary issues through an exploratory case study on a long-running, large-scale SoS: the LOFAR radio telescope and its follow-up project, LOFAR2.0.[1] These practices concern (1) how to define high-level requirements and make them properly propagate to the lower levels (e.g., as software requirements), (2) how to properly demarcate the boundaries of responsibilities at the lower levels of the system through system decomposition and high-level interfaces (3) how to organize the architecting roles of the disciplines involved, and (4) how to achieve early integration in the process.

In the current paper, we present a follow-up study that aims *to confirm and extend our previous findings* with respect to architecting practices and the cross-disciplinary issues they address. For this purpose we conduct a confirmatory case study on a ultra-large-scale SoS: the Square Kilometre Array[1] (SKA). The SKA is a global project, with eleven international consortia involved in its design, that aims to build multi-purpose low-band/mid-band radio telescopes in South Africa and Western Australia which, when working together, will collect data from an area equivalent of (at least) one square kilometre.

The overall contribution of this work can be summarized as follows. First, it confirmed in various degrees the four practices identified in the exploratory case study [18], namely (a) *Adopting a rigorous systems-engineering process where requirements are defined upfront*; (b) *Hierarchical co-architecting roles*; (c) *Splitting up the system on a subsystem level, with boundaries demarcated with Interface Control Documents (ICDs)*; and (d) *Early Integration activities/techniques*. Second, it provided further insights on these practices, in particular: (a) the friction between disciplines caused by the long-term system requirements—and hence the call for intermediate, properly traceable short-term requirements derived from them; and (b) how some prescriptive design decisions (with implications across multiple teams), and the criteria used to guide the early system decomposition have an indirect impact on the system architecture and qualities. Third, it unveiled additional and seemingly successful practices to bridge the gaps between disciplines that range from cross-level communication practices, to more technical ones oriented towards the alignment of hardware and software development cycles.

The rest of this paper is structured as follows: Section 2 summarizes the LOFAR/LOFAR2.0 study and its findings, and the particular setting of the SKA. Section 3 presents the study design. Section 4 describes the study results, and Section 5 discusses the answers to the proposed research questions based on synthesizing these results. Section 6 discusses the implications of the findings for practitioners and researchers. Finally, Sections 7 and 8 present the *Threats to Validity* and the general conclusions of the study, respectively.

2. Background

This section briefly describes the main findings of the previous case study and the characteristics of the case subject, the SKA system, from which said findings are expected to be confirmed and expanded. A full description of the former is available at [18].

2.1. LOFAR+ exploratory case study & its identified practices

LOFAR (Low Frequency Array) is one of the largest radio telescopes on Earth, and falls into the category of *directed SoS*, i.e., an SoS built and centrally managed to fulfill specific purposes [11,19]. It consists of multiple, geographically distributed stations, which in turn, operate massive antenna arrays whose readings are digitized, filtered, and transported at a central location to be processed according to the requirements of a number of scientific data products [20]. As many SoS, its development occurred in a particularly long time frame (over two decades), and involved numerous scientific and engineering challenges that led to many lessons learned. Consequently, LOFAR2.0, the ongoing expansion of the scientific and technical capabilities of LOFAR (expected to be ready in 2025), relies on the experience gathered from its predecessor [21]. Our previous exploratory case study on both projects [18] (from now on referred to as the *LOFAR+ case*) focused on characterizing the practices adopted as a consequence of this experience towards harmonizing the architecting practices of the disciplines involved in the project, and in particular between Systems Engineering (SE) and Software Engineering (SWE).

Four high-level practices were identified, namely: *Adopting a rigorous systems-engineering process where requirements are defined upfront (B1)*, *Hierarchical co-architecting roles (B2)*, *Splitting up the system on a subsystem level, with boundaries demarcated with Interface Control Documents (ICDs) (B3)*, and *Early Integration activities/techniques (B4)*. The cross-disciplinary issues identified in LOFAR and addressed in LOFAR2.0 through the aforementioned practices are related to: (a) *system requirements and how they flow down to the lower levels of the system*; (b) *local design decisions and how they may impact the overall system*; and (c) *subsystem interfaces, and how these capture the mutual expectations of the involved parties*. In the following, practices *(B1)*–*(B4)* are described in more detail, while their related issues, *(P1)*–*(P9)*, will be discussed further when discussing the study results in the following section:

(B1) In LOFAR there were issues related to the system-level requirements (also known as L1-requirements), and the way they flowed down as subsystem (e.g., software) requirements. In particular, it was perceived that in many cases the requirements were unclear or missing *(P1)*, arguably due to the fact that LOFAR was originally proposed as an instrument concept that would take advantage of a technological opportunity, where its possibilities were yet to be discovered. It was perceived that this, and other problems such as not having properly or sufficiently defined interfaces – *(P6)* and *(P7)*, respectively – was caused by the lack of a proper systems engineering process, where requirements are defined up-front (in other words, requirements front-loading). Consequently, on LOFAR2.0, this was seen as a key practice.

(B2) In LOFAR, the space for making design decisions on the subsystems (particularly the software-intensive ones) was not clearly demarcated by the system requirements *(P3)*. Given the lack of a project role that guards/drives the overall system qualities *(P5)*, this contributed, in turn, to the emergence of architectural smells from local design decisions *(P4)*. Consequently, adopting hierarchical co-architecting and co-design approaches *(B2)*, where system architects oversee both system and software subsystems/components, was seen as a key practice in LOFAR2.0.

(B3) In LOFAR, the requirements at software level were too constrained by the design decisions at system level *(P2)* as the software side was involved late in the process. Furthermore, the local design decisions under this particular setting lead to the emergence of architecture smells *(P4)*. In LOFAR2.0, it was perceived that having the system split at the subsystem level early on (instead of a hardware-, software-, and firmware-based partition), with properly defined ICDs as boundaries, was key to address these issues.

[1] https://www.astron.nl/telescopes/lofar/.

[1] https://www.skatelescope.org/the-ska-project/ The reader is advised that information about the SKA presented in this study is valid at the time of writing (November 2021). As with any ongoing project, this is potentially subject to change in the future.

(B4) In LOFAR, the integration process was considered challenging due to the substantial tweaking and trouble-shooting required. To a large extent, this is attributed to issues with the subsystem interfaces. In particular, it was perceived that such interfaces were not thoroughly thought-out *(P6)* and (arguably as a consequence of this), they did not sufficiently capture the mutual expectations between the two parties involved in them *(P7)*. For this reason, in LOFAR2.0 early integration activities/techniques *(B4)* were adopted as a means to anticipate misunderstandings, and avoid wrong assumptions from both parties of an interface (e.g. hardware and software).

It is worth noting that two more issues were identified, which were not solved when transitioning from LOFAR to LOFAR2.0. First, the integration of hardware, firmware and software was challenging given that the traditionally long development cycles of the hardware, and the short (agile) cycles of software were difficult to align *(P8)*. Second, the study revealed that the requirements were focused mostly on the steady-states of the system. That is to say, the dynamic behavior of the system and its constituents seemed to be overlooked *(P9)*.

2.2. SKA overview

The Square Kilometre Array is a hierarchical, distributed, ultra-large-scale next-generation radio telescope, which entered its construction phase in June of 2021. The construction budget for this phase of the telescope is around 700 M€. Scientific operations of the first phase of this instrument, consisting of a 197 dish mid-frequency array (including 64 existing MeerKAT dishes) in the South African Karoo desert and a 130.000 element low-frequency phased array in Western Australia, is expected to start sometime around 2027. Total construction of the first phase of SKA, including contingency and labor, will cost around 1.28 billion €.[2]

The SKA construction, commissioning and exploitation is governed by the SKAO, the SKA Observatory, a non-governmental organisation (NGO) that was established in March 2019 and came into effect in January 2021. The SKA design was conceived during a protracted design phase, which saw contributions from sixteen countries and around a hundred different organizations. While no public estimate exists on the number of engineers involved in this process, project-wide engineering meetings consistently drew between two and three hundred engineers.

2.2.1. SKA high-level architecture

The architecture of the SKA involves two large-scale subsystems, namely the Mid and Low radio-telescopes, which capture low and mid frequencies in the electromagnetic spectrum through arrays of dipole antennas and arrays of dish observatories. As described in Fig. 1, both subsystems have equivalent architectures, where the raw data is captured through the respective arrays and turned into *visibilities* (pulsar survey candidates and their timing) through the *Central Signal Processor* (CSP). The vast amount of data generated by the CSP is ingested, in turn, by the *Science Data Processor* (SDP), where data is reduced and packed so scientists can make decisions in near real-time, e.g., about noise in the captured data.

Both of these large-scale subsystems consist of components that are either Systems Engineering dominated, Software Engineering dominated, or both. While the SKA will be much more software-focused than previous generations of radio telescopes, we can generally assume that a software component becomes more pronounced as we move along the processing chain. In other words, receivers are heavily dominated by Systems Engineering with a small software component for local monitoring and control, a system such as the Correlator and Beamformer is a mix of Systems Engineering for board design and Software Engineering for FPGA firmware development, while the Science Data Processor and Telescope Monitoring and Control are dominated by Software Engineering, as shown in Fig. 1.

2.2.2. SKA architecting and development process

During its pre-construction phase, the system was partitioned in multiple subsystems and then delegated to eleven international consortia for their detailed design, using ICDs to have a clear separation of responsibilities (from now on, this will be referred as the *early partition of the system*). In the SE process followed in this phase, the compliance of the design against the Subsystem Level (L2) requirements (derived from the System Level (L1) ones in this phase) and the ICDs were rigorously supervised. This phase also involved a number of preliminary design reviews of the architecture and was closed with a formal Critical Design Review (CDR), where the final design of each subsystem is presented through completed analysis, simulations, schematics, software and test results [22].

The transition between the CDRs and the construction phase, which started at late 2019 and is coming to a closure as this paper is written, is described as the *bridging* phase. Interestingly, during this bridging phase, the organization opted to adopt the *Scaled Agile Framework* (SAFe) [23] approach to tackle the challenge of developing the software-intensive elements aligned to a common vision, after the pre-construction consortia got dissolved. This involved, in general (1) forming an *Agile Release Train* (ART), that is to say, cross-disciplinary agile teams that will work towards such a common vision, (2) defining and prioritizing a backlog of features for each *Program Increment* (PI), which are development sprints planned by the aforementioned teams during the *PI-planning* meetings, and (3) driving each team work following a *Scrum* process. The SKA is a unique case, among systems of this scale, where traditional SE practices (particularly the ones followed in the pre-construction phase), meet with the agile principles and the practices described above. This means, for instance, that the requirements gathered during the consortia phase (both L1 and L2) were used as an input for defining the *Solution Intent (SI)*, an evolving knowledge base that serves as the 'single source of truth' about the product vision, what is being built, and how it will be built. This transition from SE to an agile process also implies the need for managing two types of interfaces: the high-level ones originally defined through the ICDs, and the low-level ones created within the cross-disciplinary teams, and specified within the Solution Intent.

3. Study design

3.1. Study goals and research questions

The goal of this study, as a follow-up of the exploratory case study on LOFAR+, is to *confirm and extend the practices that harmonize architecting processes of Systems Engineering (SE) and Software Engineering (SWE) in large-scale SoS*. A more precise definition of this goal, using the five parameters proposed by the Goal–Question–Metric (GQM) goal template [24] is as follows:

> **Analyze** *the architecting process of an ultra-large-scale SoS involving multiple organizations* **for the purpose** *of confirming the practices for harmonizing SE and SWE architecting processes identified in the context of the LOFAR+ case study, and identifying new ones* **with respect to** *cross-disciplinary issues related to requirements, subsystem design decisions, and subsystem interfacing* **from the viewpoint of** *engineers*, **in the context of** *the Square Kilometre Array (SKA) project.*

Based on this goal, we derived the following research questions:

[2] SKA Phase 1 construction proposal (https://www.skatelescope.org/key-documents/).

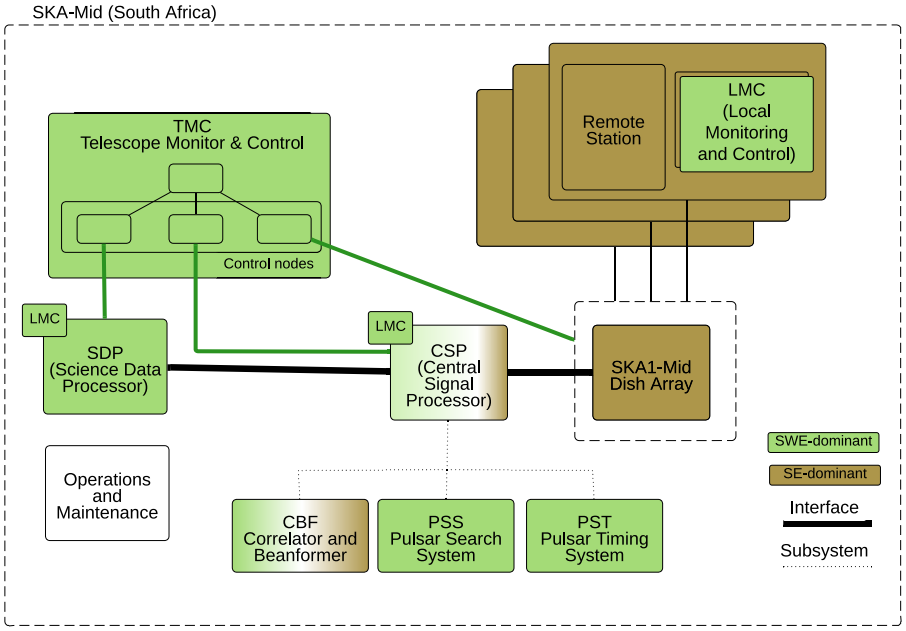


Fig. 1. Simplified model of the SKA system architecture, including the key components for its operation. Only the SKA-Mid subsystem is presented here. The SKA-Low radio telescope subsystem architecture is similar to it but with a low-frequency aperture array (arrays of dipole antennas) instead of dish arrays.

RQ1. *To what extent have the harmonization practices identified in LOFAR+, successfully addressed the same kind of issues in the context of the SKA, on requirements (RQ1.1), subsystem design decisions (RQ1.2), and subsystem interfacing (RQ1.3)?*

RQ2. *What additional cross-disciplinary harmonization practices, related to requirements (RQ2.1), subsystem design decisions (RQ2.2) and subsystem interfacing (RQ2.3), were adopted in SKA? What additional practices were adopted to address the issues that remained un-solved in LOFAR+ (RQ2.4)?*

Along the same lines of the exploratory case study [18], whose results this study aims to confirm and extend, these research questions entail investigating them within its real-life context. Consequently, we opt for a *confirmatory case study* as an empirical research method [25].

Table 1
Mapping between research and survey questions, including the issues and best practices explored per question.

RQ	Scope	Survey questions	LOFAR+ Practices
N/A	Demographics	Q1.1–Q1.7	
RQ1.1 RQ2.1	Requirements Requirements	Q2.1–Q2.4, Q2.6 Q2.7	*(B1)*
RQ1.2 RQ2.2	Design decisions Design decisions	Q3.1–Q3.5 Q3.6, Q3.7	*(B2)*,*(B3)*
RQ1.3 RQ2.3	Interfacing Interfacing	Q4.1, Q4.2–Q4.5 Q4.5	*(B4)*
RQ2.4	Requirements and Interfacing (issues not addressed in the LOFAR+ study)	Q2.5, Q4.3, Q4.6, Q4.7	

3.2. Data collection

The SKA, as described in the previous section, is a massive project that involves hundreds of engineers around the world. To better understand the case subject (i.e. the SKA project) and to define properly tailored research instruments, we conducted an *in-situ* pre-study by analyzing relevant archival data of the project, e.g., the current (at the time) Solution Intent document, and most importantly, by participating as observers on the 10th PI-planning event (taking place virtually due to the COVID-19 pandemic). The former allowed us to gain an up-to-date understanding of the principles followed by the SKA for its evolving design and implementation. The latter helped us gather details (also related to such principles) about the early stages of the SKA, and the way requirements, design decisions and interfaces were managed in the project. The event's onboarding sessions, where questions were proposed and discussed as relevant topics arose, were particularly useful for this purpose. Separate meetings with the SAFe consultant of the project also served as a source of information about the transition of the project into this framework.

Based on the findings of this pre-study, we designed a *questionnaire-based survey* for data collection purposes. The survey instrument was tailored to the target participants, that is to say, considering the specifics of the SKA context and the terminology used by the engineers involved in the Agile Release Train. As described in Table 1, the survey questions were organized in four sections: *demographic data*, *requirements-related issues and practices—(B1)*, *design decisions-related issues and practices—(B2)* and *(B3)*, and *interface-related issues and practices—(B4)*. Questions concerning the handling of issues identified in LOFAR+ without best practices to address them, i.e. *(P8)* and *(P9)*, were added to the survey sections relevant to them, namely the second and fourth sections of the survey.

We used a combination of closed and open-ended questions in order to achieve in-depth insights. Most of the closed questions used a five-point Likert scale to measure perceptions pertaining to, for example, the impact of a given practice in the context of the SKA. Given the broad range of elements addressed by the investigated practices (requirements, design decisions, interfacing), and the low likelihood as we perceived it of all respondents to be familiar with all of them (especially due to the system scale), an explicit 'Do Not Know' option was added as a sixth option in the answer scales. This allowed to avoid mixing *no opinion* responses with *neutral/low-score* ones.

The survey was piloted by three ASTRON engineers that also participated on the LOFAR+ case study, and were familiar with the SKA project by being partially involved on it. These three participants were not included in the sample population of the survey, and neither was any of the other LOFAR+ study participants. With the input received from the pilot's participants, the structure, wording and the use of the

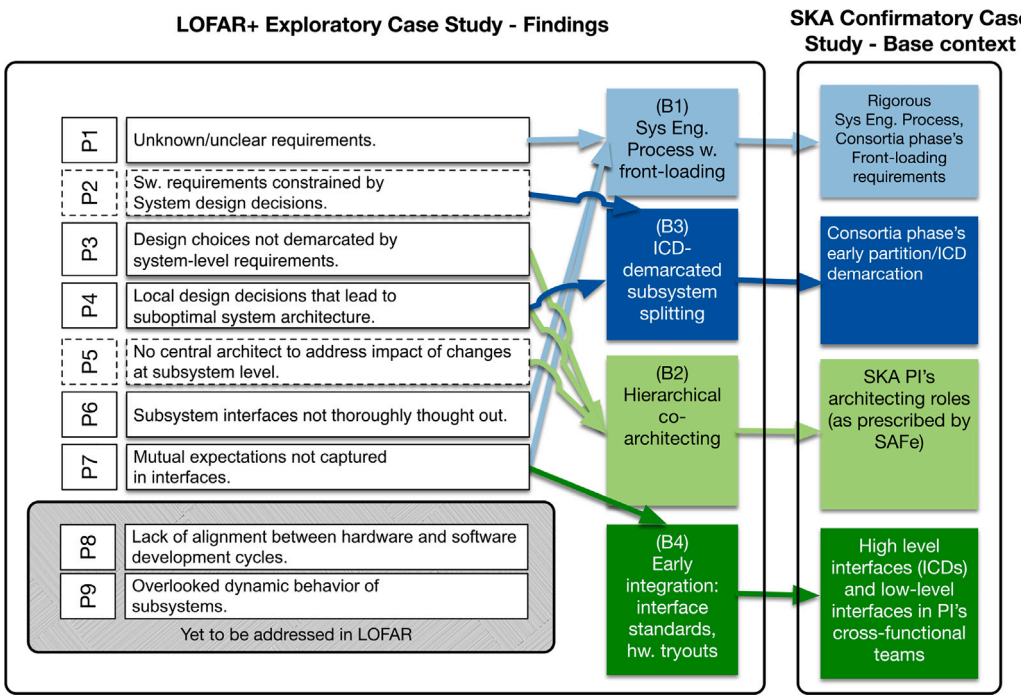


Fig. 2. Summary of the SKA practices that reflect the ones identified in the LOFAR+ case, as identified by the pre-study analysis results.

terminology were improved for clarity. The survey was later reviewed by one of the SKA's lead software architects, who also contributed to its improvement. Furthermore, the survey activity was validated by the SKA Organization's head of Mission Assurance. The final version of the survey was developed and published online through the Qualtrics.XM platform.[3] The target population, which consisted of SKA engineers that work under the SAFe framework, was profiled with the help of the head software architect, who estimated it in around 140 people. He also customized a mailing list to match the target population and used it to submit invitations to participate. This was followed by a number of invitations through the SKA's Slack group, and through personal interventions in activities within the 10th PI, as discussed above.

The final form of the questionnaire is available in the study replication package.[4]

3.3. Data analysis

The data collected from the survey were analyzed using descriptive statistics for the closed questions, and Qualitative Content Analysis (QCA) for the open-ended ones. In the latter, an inductive approach was followed [26], meaning that the identified categories and groups of codes emerged from a systematic process of open-coding, categories creation, and abstraction. The most prominent categories together with representative quotes can be found in the online Appendix to this paper, included together with the full data in the replication package of this study (see Footnote 4). The results of the data analysis are discussed in the following section.

4. Results

In the following, we present our analysis of the survey responses, which will be subsequently used in Section 5 to answer the proposed research questions. To this effect, the presentation follows the structure of the survey, i.e. discussing demographic data, requirements-related issues and practices, design decisions-related issues and practices, interface-related issues and practices, and practices for the issues identified as not solved in the LOFAR+ case. With the exception of the subsections that describe the demographics and the practices for unsolved issues on LOFAR+, each of the other subsections are introduced with a description of the specific practices within the SKA that reflect the ones defined in the LOFAR+ case. This mapping between SKA and LOFAR+ practices, which resulted from the pre-study conducted based on Observational and Archival data as described in the previous section, is summarized in Fig. 2.

4.1. Demographics

A total of 46 responses were collected between late April and early September of 2021, out of the total target population of around 140 as discussed in Section 3. The base organizations of the respondents are distributed across the globe as illustrated in Fig. 3. The respondents have an average 18.13 years of work experience, with 5.6 years of experience in the SKA project. As seen in Fig. 4, most of the respondents identify themselves with multiple roles within the project, with *Software Developer* (20%), *Domain Expert* (15%), *Product Owner* (14%) and *Software Architect* (12%) being the most prominent ones.

Project coverage

As illustrated in Table 2, all SKA subsystems being either Software Engineering dominated or mixed, are represented by the respondents to this survey. This is sufficient for purposes of ensuring the representation of all relevant components as we are conducting this study from a clear Software Engineering perspective.

Opt-outs

The information collected by the demographics section of the survey provides insights on the opted-out (i.e. 'Do Not Know') responses to closed questions (Likert scale) in the rest of the survey; on average 12.65 participants opted out per closed question. In particular, by running Spearmann correlation tests between the respondents' number of opted-out responses and their years of overall and SKA-specific

[3] https://www.qualtrics.com/.
[4] https://doi.org/10.6084/m9.figshare.17040362.v1.

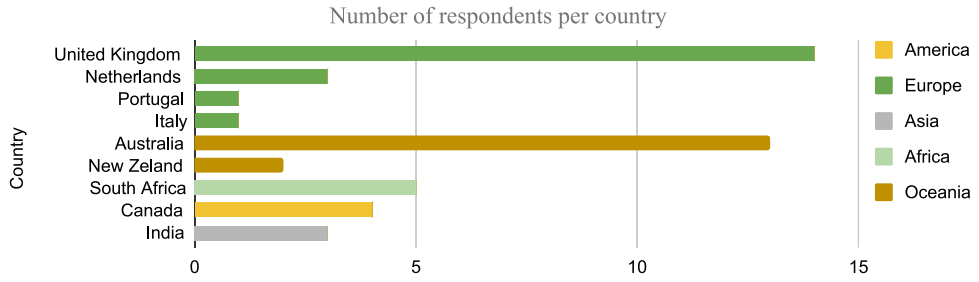


Fig. 3. Distribution of the respondent's organizations across the world.

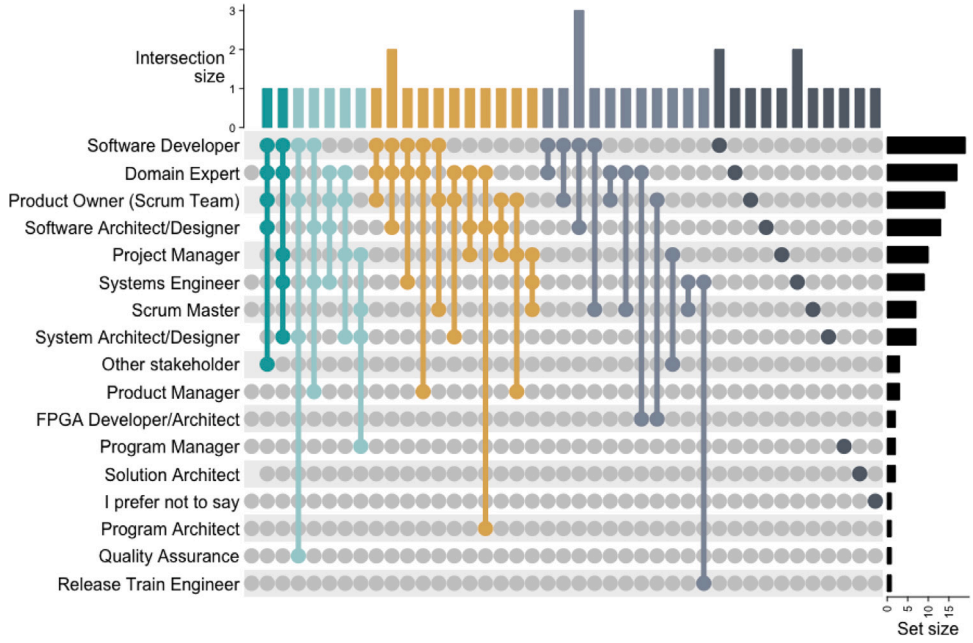


Fig. 4. Roles performed in the SKA by the survey respondents.

Table 2
Distribution of the interviewees in terms of which components of the system architecture (described in Section 2 & Fig. 1) they work on, given the teams they belong to. The bottom half of the table lists the participants whose teams work on/provide support to artifacts concerned with multiple subsystems within the architecture.

Subsystem	Participants
SDP	13
TMC & CSP	1
TMC & LMC	4
CSP	4
TMC	7
Observation management and controls	3
Data processing/Quality metrics	3
Services support	2
Assembly, integration and verification	2
Not disclosed	7

experience, we found only a slight negative correlation for the SKA experience (rho = −0.32 with a p-Value of 0.032) and weak evidence for the overall experience (rho = −0.14 with a p-Value of 0.35). Therefore, in general, experience levels do not sufficiently explain the number of opted-out responses presented in the following sections. However, by grouping the roles in more general ones,[5] namely *Architects*, *Engineers/Developers*, *Managers*, and *Others*, and counting the opted-out questions by each group, we found that most opt-outs came from two groups: managers, and engineers/developers as summarized in Table 3. Arguably, this could be attributed to two different phenomena: for the managers, due to the lack of sufficient depth of technical details that some of the questions required; for the engineers/developers, due to the very specific roles within the project performed by this part of the population allowing them to focus only on certain parts of the survey. Both explanations are consistent with our decision of including an explicit 'Do not know' option on the Likert scales, as described in Section 3.2.

4.2. Practices and issues related to requirements

In the LOFAR+ case, the issues of *unclear requirements (P1)*, *interfaces that are not thoroughly thought out (P6)* or that *do not capture*

[5] **Architects**: Solution Architect, Program Architect, System Architect/Designer, Software Architect/Designer; **Engineers/Developers:** Systems Engineer, Software Developer, Scrum Master, Quality Assurance, Product Owner; **Managers**: Project Manager, Program Manager, Release Train Engineer; **Others:** Other roles, Other Stakeholder, *I prefer not to say*, Domain Expert.

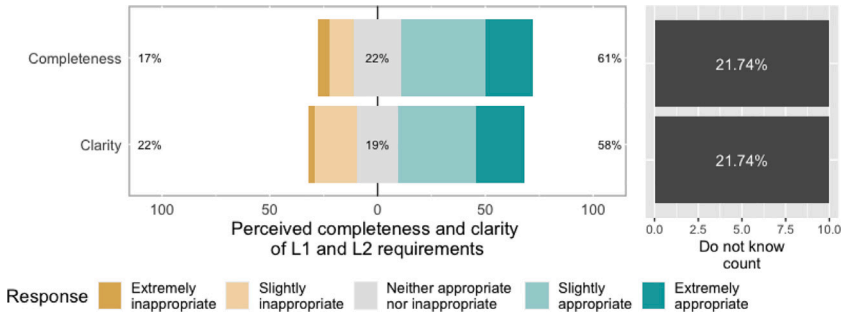


Fig. 5. Distribution of the responses to Q2.1 about the perception of clarity/completeness of L1/L2 requirements in the SKA project, using as a diverging point the neutral opinion. The opt-out selection (shaded) does not belong to the scale but is included as a reference.

Table 3
Mean and standard deviation of the number of 'Do Not Know' answers selected across the 28 Likert questions of the survey by groups of roles, normalized by the corresponding groups size.

Role group	Group size	Total opt-outs	$\bar{x}$	σ
Managers	12	93	0.28	0.12
Engineers/developers	35	269	0.27	0.07
Architects	20	122	0.22	0.09
Other roles	23	133	0.21	0.11

the mutual expectations of their related parties (P7) were addressed by adopting a *more rigorous systems engineering process, with front-loaded requirements (B1)*. As depicted in Fig. 2, this practice can be seen as already followed in the SKA, as the project defined and allocated up-front, in the consortia phase described in Section 2.2, a series of System Requirements (known as L1 requirements) and Element (or sub-system) requirements (known as L2 requirements) [27]. However, for the SKA construction, part of the software-intensive elements of the SKA are being developed as a Solution Intent, following SAFe, which encourages leaving room for an emerging understanding of specific requirements, based on intent. Given this, we explore practice *(B1)*, in the context of the SKA, by investigating how these front-loaded requirements (i.e., L1 and L2) helped or hindered the Solution Intent. Furthermore, we investigate what additional practices have been adopted by the SKA to address requirements related issues, including *(P1)*, *(P6)* and *(P7)*.

According to the results shown in Fig. 5, and using a neutral opinion as the diverging point, most of the respondents (excluding the ones that opted out of the question) evaluated the completeness and clarity of L1 and L2 requirements *(P1)* as appropriate or extremely appropriate (61% and 58% respectively). However, when looking at the degree to which participants have had to make assumptions while working with these requirements (see Fig. 6), and to which these were used in the early tests design (see Fig. 7) as two of the consequences of having incomplete/unclear requirements according to the LOFAR+ case, this seems rather counter-intuitive. As seen in Fig. 7, despite the perceived clarity and completeness of the requirements, around 85% of the respondents (excluding the ones that opted out of the question) feel that L1/L2 requirements had a moderate to no influence on the design of tests early in the development process. Furthermore, as described in Fig. 6, most of the respondents perceive that working with this kind of requirements involves making assumptions to a moderate extent, especially regarding system-level (L1) requirements (62%). Given the positive perception of the clarity and completeness of the requirements defined up-front *(B1)*, the results regarding the need for making assumptions when working with system-level requirements, and their influence of early testing, were not expected. We therefore look into the open-ended questions for a more nuanced analysis of the practitioners' perceptions.

The analysis of the 38 open-ended responses to Question 2.6, about the perception of the positive or negative impact of these front-loaded requirements *(B1)* in the Solution Intent, shows a predominant positive view of them as helpful, albeit to a different extent (codes with a total groundedness[6] g of 36). There is also a smaller minority with a negative view due to a perceived lack of actual influence of said requirements in the Solution Intent (codes with $g = 5$). Both points of view are described in the following.

4.2.1. Views that confirm the practice

According to the quotes assigned to the emergent categories as Helpful in General (g=10), Solid Ground (g=7) and Solution Intent Base (g=4) as described in Table A.1 (in the online Appendix—see Section 3.3), L1/L2 requirements are key for ultra-large-scale, long-term projects like the SKA as they provide an initial partitioning criteria, making them more manageable *(P6)*. Furthermore, the requirements provide vital elements that should flow down to the system design and to the Solution Intent; in particular these elements are edge cases, data formats, and 'nice-to-have' features *(P1)*. More importantly, though, the requirements provide information in advance regarding formal testing *(P1)*, although this does not seem to be always the case as shown in Fig. 7.

The responses coded as Transition to Agile Issues (g=5), on the other hand, highlight that L1 and L2 requirements while helpful are also in danger of being inconsistent as the Solution Intent evolves, and (eventually) they will clash with reality once the instrument is built. For this reason, there is a perceived danger of losing the focus during the agile process as the requirements are not serving as the framework to govern the development in the long term. Respondents also highlight the need to fine-tune this transition from front-loaded requirements to Program Increment: they mention that, in retrospect, moving from requirements-driven to an agile development approach made the project slow down, and that L1 requirements could have been defined (if such an approach switch were foreseen) in a way that suits better the adopted agile framework.

Similarly, the responses with the Missing Intermediate Elements (g=6) code pointed out the need for having additional intermediate elements between the front-loaded requirements and the agile process. It is perceived that the project requires better traceability between L1/L2/L3 requirements (where L3 requirements refer to implementation specifics), and the features/tasks the agile teams are working on. According to the respondents, this would be useful to identify if the understanding of the system – the one that emerges from the agile process – is deviating from the original system definition. At the same time, and given the very long term of the project, it is perceived that not having requirements for intermediate releases (in other words, short- and medium-term requirements) hinders the Solution Intent. According

[6] The groundedness g of a code refers to the number of quotations extracted during the coding phase that ended being associated with it through the inductive Qualitative Content Analysis process (from which such a code emerged) [28].

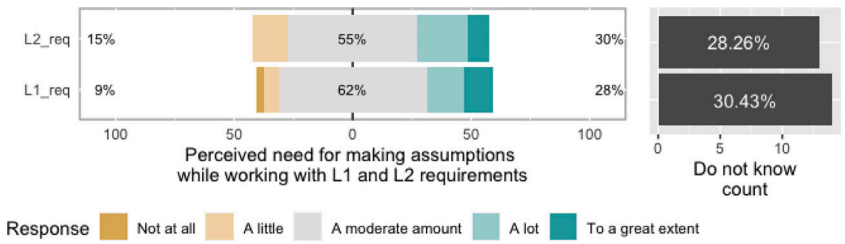


Fig. 6. Distribution of the responses to Q2.3 about the extent to what the participants and their teams had to make assumptions with the requirements elicited from the Systems Engineering process.

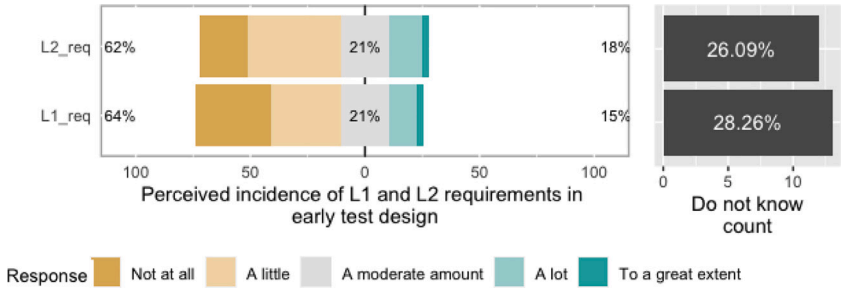


Fig. 7. Distribution of the responses of Q2.2 about the extent to what L1 and L2 requirements have been used for early tests design.

to the respondents, *requirements for the intermediate versions of the system are key* to properly drive (accordingly to the final solution vision) the architecture, and for having proper test specifications of the software systems. Furthermore, it is perceived that the long-term requirements do not always capture the activities needed for specific intermediate phases such as construction and commissioning.

Finally, the code Imperfect Req Issues (g=3) groups the comments where respondents pointed out that L1 and L2 requirements, when not well defined, result in constant refactoring. Some respondents describe this as requirements with a lot of elements left to be assumed or discovered. It was pointed out that, for a proper requirements tree, an iterative validation process of the flown-down requirements completeness, i.e., from L1 down to L2 and L3, and all the way back up, would be necessary.

4.2.2. Views that do not confirm the practice

According to the respondents whose responses were coded as Little Weight (g=2) or Not Followed (g=3), as described in Table A.2, L1/L2 requirements have little weight or are ignored during decision making in particular cases. Respondents mentioned two perceived causes for this: this kind of requirements are not useful for a lean/agile process, or in some cases teams do not follow them due to budget and time constraints. Other respondents mentioned – arguably as a consequence of the above – that the Solution Intent (at the moment of filling in the survey) was driven more by assumptions and experience of the architects rather than the L1/L2 requirements.

4.2.3. Additional practices adopted by SKA to address requirements-related issues

According to the analysis of the 31 open-ended responses registered for Question 2.7 about how requirements-related issues like unclear and incomplete requirements are being prevented or mitigated in the SKA, the practices followed in this direction are mostly mitigation-oriented. A number of respondents, as seen in code Agile Filling Gaps (g=9) (see Table A.4), pointed to SAFe as the key strategy to mitigate incomplete or unclear requirements given the realization that L1 and L2 requirements inevitably evolve over time, and that in an agile framework like SAFe, evolving requirements are expected to be part of the basic understanding of the system. In the same question, other respondents (code Prototypes Testing (g=5)) see test-driven design as a key agile practice to identify requirement issues as the construction progress. This has been perceived at two different stages: early during the definition of the acceptance tests, and while receiving feedback from stakeholders (often involved in the definition of the original requirements) through prototype testing and demos. A few respondents, on the other hand, highlighted that the SKA already mitigates these issues with significant efforts on requirement reviews, where traceability, consistency and completeness are evaluated, and assumptions documented (Reqs Reviews (g=4)).

4.3. Practices and issues related to subsystem design decisions

In LOFAR+, *local design decisions had a negative impact on the overall system qualities and architecture (P4)*, as such decisions were *not properly demarcated (P3)*, and due to the *lack of a central architect role (P5)*. Proper subsystems splitting and demarcation with ICDs *(B3)* and having hierarchical co-architecting roles *(B2)* were two practices identified as key to address these issues.

These two practices can be seen as already applied on the SKA, as (1) during its consortia phase (see Section 2.2) the system was subject to such a partition/ICD demarcation, with the resulting architecture being the baseline of the project's Solution Intent, and (2) the adopted SAFe framework prescribes such hierarchical co-architecting roles (solution/system architects). Given this, as also depicted in Fig. 2, we explore practices *(B3)* and *(B2)* in the context of the SKA by investigating the perceived impact of this early partition, and the hierarchical architecting roles on the Solution Intent, on the system architecture and qualities. Furthermore, we investigate further practices adopted by the SKA to address design decisions-related issues, including the use of non-functional requirements (NFR) as the means prescribed by the SAFe framework to properly demarcate the design decisions *(P3)*.

As it can be seen in Fig. 8, the negative impact of the local design decisions on the system design and its qualities is perceived, for the most part, as small to moderate. This suggests that the SKA is seemingly doing a good job when it comes to this particular issue previously identified in the context of LOFAR+ *(P4)*. However, given the perceived contribution of the early partition of the system *(B3)*, and the hierarchical architecting roles *(B2)* to mitigate issues related to local design decisions (described in Figs. 9 and 10 respectively), the latter seemed

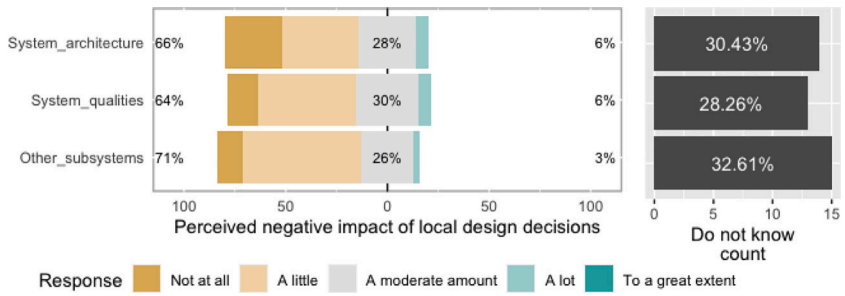


Fig. 8. Distribution of the responses of Q3.1 about the extent to what local design decisions have had a negative impact on other subsystems, system-level qualities or the overall system architecture.

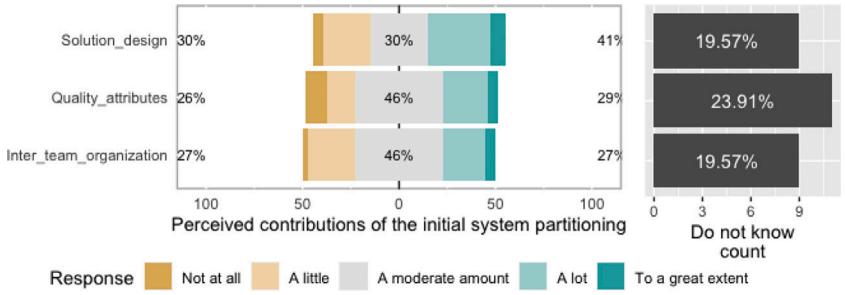


Fig. 9. Distribution of the responses of Q3.4 about the extent to what the early partition of the system has contributed to the overall solution design, the system qualities and the inter-team organization.

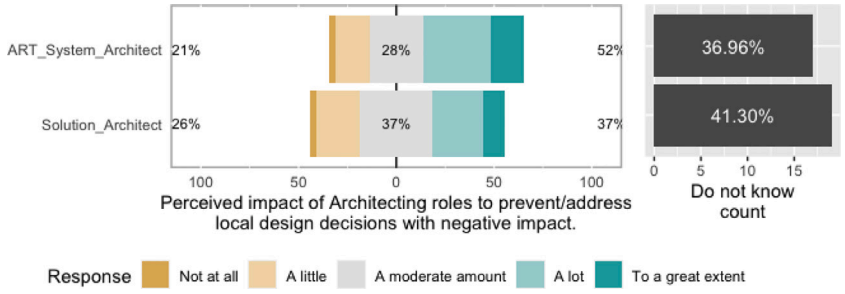


Fig. 10. Distribution of the responses of Q3.5 about the extent to what the hierarchical architecting roles prescribed by SAFe contribute to prevent or address the negative impact of local design decisions on the overall system qualities.

to have a more significant influence in the positive perceptions on that respect.

As a result of the analysis of the 28 open-ended responses recorded for Question 3.2, two main categories of design decisions (Table A.5), were perceived as having a negative impact on the overall system design and its qualities . The first and one of the most dense ones, Prescriptive Decisions (g=8), corresponds to a number of *technical, often prescriptive cross-cutting design decisions* with a perceived negative impact on the system due to the constraints or limitations they posed when implementing the related subsystems. Table 4 lists a summary of specific design decisions of the aforementioned category, as described by the survey participants (the referenced subsystems are described in more detail in Section 2.2). It is worth highlighting that some of these design decisions, which contributed to an overly-complicated architecture, are linked to the initial partition of the SKA – related to Practice *(B3)* – as its root cause, e.g., the seemingly unnecessary abstraction layers.

The quotes of the second most prominent code (Local Interpretations (g=8)), pointed out that some decisions, when based on the team member's previous knowledge on what makes more sense to them, or on their own interpretation of the requirements, have had an impact on qualities like availability, maintainability, and flexibility. In addition, this kind of decisions often leads to extra effort during integration, e.g., when integrating subsystem components that are originally operating on their own local networks into a single, large network. From the responses related to this code, it is worth noting that: (1) these are closer to the issues identified in the LOFAR+ case *(P4)*, where design decisions made independently at the subsystem level have an impact on the system architecture and its qualities; and (2) they contradict, to some extent, the point of view of the comments of the previous category, where some early, prescriptive decisions were an issue, as they highlighted that having a holistic view of the system defined early on would have helped to make better design decisions.

4.3.1. Additional practices adopted by the SKA to address subsystem design decisions-related issues

Most of the quotes extracted from the 26 open-ended responses to Question 3.6 (Table A.6) about the practices adopted by the SKA to prevent local design decisions having a negative impact on the overall system architecture/qualities *(P4)*, were classified as communication-oriented ones (Communi-cation Related (g=9)). It needs to be noted however that such practices are still seen as a work in progress. The relevant quotes agree on the benefits of enabling higher levels of transparency within the project through *active communication across levels* (e.g., to get input regarding potential impact of the design decisions), and by *facilitating access to key roles and groups*, namely systems engineers,

Table 4
Summary of the 'prescriptive', cross-cutting design decisions that had a perceived negative impact on the system design and its qualities. Subsystem names refer to Fig. 1.

Design decision	Consequence
Algorithmic prescriptions on the signal processing pipeline (e.g., filter coefficients)	Global signal processing optimization alternatives limited.
Delegating computation responsibilities at a certain system level (e.g., delay calculations)	Workarounds to overcome the limitations of the selected system level.
Data (visibilities) resolution	Mismatch between the expected resolution and the limitations of the devices
Transport Protocol selection (e.g., between the CSP and the SDP)	Interfacing/Performance issues
Additional abstraction layers for integration (e.g., for the TCP)	Over-complicated design (e.g., unnecessary pass-through nodes in the system)
Use of common base-classes across sub-systems	Broken deployments and large maintenance efforts when such classes are changed.
Deployment infrastructure selection (e.g., orchestrated containers for both TMC and SDP)	Compatibility/performance issues

solution architects, architecture groups and communities of practice. From this series of responses, it is worth highlighting that some of these communication-oriented practices emerged as a measure for the perceived risks of the 'siloed' architecture: local design decisions that could be made without considering the impact in the other subsystems *(P4)*. This, on the other hand, is consistent with the perception of the respondents about the influence of the hierarchical architecture *(B2)* within the SAFe process as previously discussed: according to the respondents, this risk has been mitigated with architecting roles, e.g., working towards consensus-driven common solutions by engaging the architectural experts on both sides of the ICDs.

4.4. Practices and issues related to subsystem interfacing

In the LOFAR+ case study, the interfacing between subsystems was identified as a significant source of issues for the interplay between different engineering disciplines, and in particular on how their documentations were not capturing the mutual expectations of the involved parties *(P7)*. To address these issues, early integration practices *(B4)* were identified. In this case, the pre-study did not provide enough information on which specific early integration practices have been followed in the SKA. Therefore, to explore *(B4)* in the context of the SKA, we investigated to what extent the interfaces within the SKA are subject to *(P7)*, and which particular practices have been followed to address such an issue. We consider both high-level and low-level interfaces in the project (as depicted in Fig. 2): the ICDs defined during the project's consortia phase, and the interfaces used within the ART's cross-functional agile teams. Furthermore, for the former, we take into account that in the SKA, ICDs were refined through a number of formal design reviews, as prescribed by the systems engineering practices [29].

The perception of the extent to which the ICDs in SKA captured the mutual expectations of the involved parties, captured in Question 4.1 and illustrated in Fig. 11, suggests two things. First, the *Critical Design Review* had a seemingly significant impact on this particular aspect of the ICDs, and second, the overall perception of the ICDs – regarding how well they captured mutual expectations after said formal reviews – is mostly positive. Regarding the lower-level, internal interfaces, as seen in Fig. 12, software-to-software interfaces either within the same agile team, or between separate ones, have a highly positive perception when it comes to the way they capture mutual expectations. For hardware/software interfaces, the perception (with exception of internal ones) is significantly more negative (mostly little to moderate capture of mutual expectations) regarding interfaces used by separate teams.

Most of the codes extracted from the 29 responses given to Question 4.2, about *what was missing on the ICDs to properly capture the mutual expectations of the involved parties*, are related to either missing elements on the ICDs (multiple codes with a total groundedness of 14), or the ICDs' life-cycle (with a single code of $g = 7$), as listed in Table A.7. In the former, four quotes (Incomplete Cases (g=4)) mentioned specific examples of incomplete or vague ICDs within the SKA (e.g., TM to LFAA,[7] TMC[8] to Elements, CSP's LMC[9] to sub-elements—see Fig. 1) without going into specifics. The other ten quotes (coded as Missing Elements (g=10)) pointed to the need of further context on how the subsystems related to an ICD would operate within the overall system, and an end-to-end picture of how things will work. In particular, they mentioned as elements to provide such a context: (1) time- and state-behavioral details, e.g., what will be passed between the subsystems through time, the states in which operations can be performed, and how the failures/errors should be handled; (2) traceability to the original requirements; and (3) details about the technology that will be used on the production environment, so more proper tooling can be selected for the development environment early on. Regarding (1), it is worth mentioning that the lack of details about the dynamic behavior of the system was also identified as an issue yet to be addressed in the LOFAR+ case *(P9)*.

The quotes coded as Life Cycle Related (g=7), on the other hand, highlighted that the initial partition of the system was mostly guided by consortium divisions (which was necessary due to the scale of the project, as described in Section 2.2) rather than logical products, with the ICDs for demarcating responsibilities between the corresponding subsystems *(B3)*. According to some of these respondents, this made the boundaries defined by the consortia, and the boundaries of the existing observatories incongruent (i.e., the partition was defined by responsibilities, not by the systems per se). Because of this, and in addition to the overly complicated architecture mentioned in Section 4.3, this led to the emergence of unofficial, internal ICDs, that are in some cases not documented. Furthermore, it was pointed out that: (1) the ICDs were often developed by people without enough experience, and not supported by prototyping/up-front design; (2) not all parties were equally involved, hence the input level was not the same; and (3) ICDs have not been properly updated as the design progressed after the consortia phase. The multiple-choice Question 4.5 concerned the *practices adopted by the SKA to address the issues of interfaces that do not capture the mutual expectations of the involved parties (P7)*. Among the responses (as presented in Fig. 13), which included the early-integration practices identified in the LOFAR+ case, early hardware/software tryouts *(B4)* is the more recognized one. It is followed by the formal documents (ICDs), hardware simulations, and other informal interface specifications with a similar number of selections. As additional practices to address this issue, from the seven additional entries given to the 'Other' option, it is worth highlighting one mentioned twice: the adoption of a semiconductor industry practice, namely *the use of a machine-readable formalism as a 'single source of truth' for the interfaces within the teams*, and a related workflow for the *automatic generation of key components for both sides of the interface* (e.g., interfaces and documentation). In the same spirit, having a live, constantly updated documentation (one of the goals of the aforementioned workflow), was also included as a key practice. Finally, the practice of *interdisciplinary collaboration*, also added by the respondents as 'Other' practice, is consistent with Fig. 12, as hardware/software interfaces within cross-functional teams seem to have significantly fewer issues regarding capturing mutual expectations in the interfaces.

[7] Telescope Manager to Low Frequency Aperture Array.
[8] Telescope Management and Control.
[9] Central Signal Processor's Local Monitor & Control.

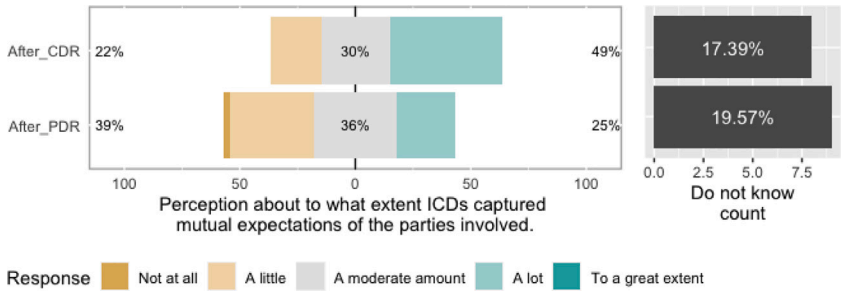


Fig. 11. Distribution of the responses of Q4.1 about the extent to what interfaces defined in the early partition of the system captured the mutual expectations of the parties involved.

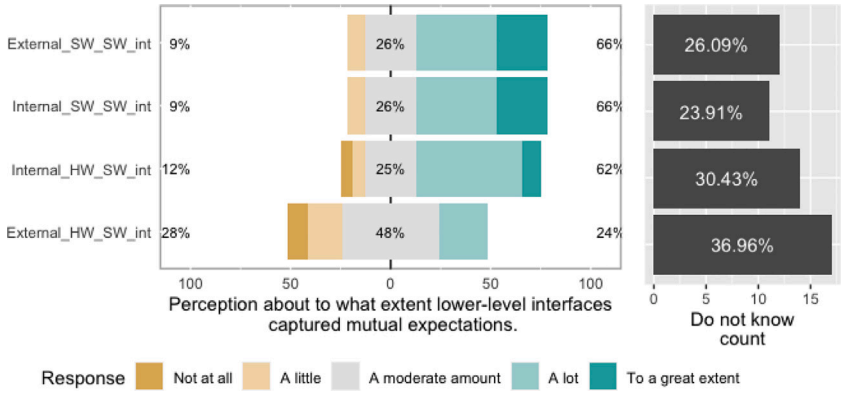


Fig. 12. Distribution of the responses of Q4.4 about the extent to what lower-level interfaces, within and between cross-functional teams captured the mutual expectations of the parties involved.

Fig. 13. Responses to Q4.5 about the early integration practices adopted in the SKA to avoid or mitigate interfacing-related issues: incomplete interfaces or interfaces that do not capture mutual expectations. The new practices reported by respondents include an '*'.

4.5. Issues still under exploration in LOFAR+ in the context of the SKA

In the LOFAR case, the *lack of alignment between hardware and software development cycles (P8)*, and the *missing details about the dynamic behavior of the system (P9)* in both requirements and ICDs, were identified as yet to be addressed issues that hindered the interplay between SE and SWE architecting processes (gray area in Fig. 2). According to the insights about problem *(P9)* discussed in Section 4.4, and the perception of the respondents on the extent to which dynamic/time-behavioral details are given in ICDs and system requirements (Figs. 14 and 15), this seems to be also an important, yet un-addressed issue in the SKA, especially with respect to requirements.

When it comes to Issue *(P8)*, on the other hand, the perception about the extent to which the development cycles are misaligned in the

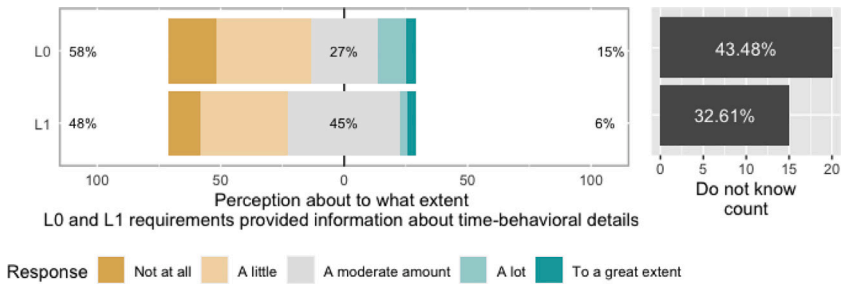


Fig. 14. Distribution of the responses of Q2.5 about the perception of the extent that L0 and L1 requirements provided time-behavioral related information.

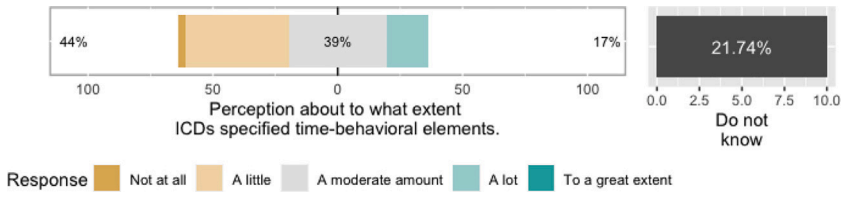


Fig. 15. Distribution of the responses of Q4.3 about the extent to which ICDs captured details about the dynamic behavior of the involved subsystems.

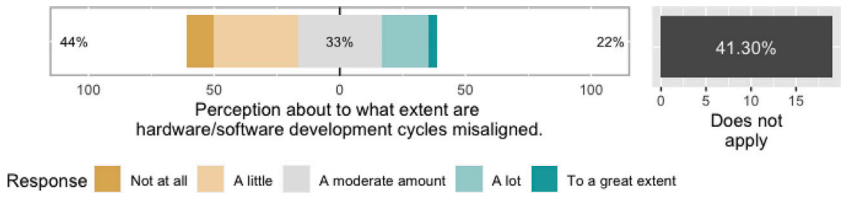


Fig. 16. Distribution of the responses of Q4.6 about the extent to which – if the respondent has been involved in hybrid hardware/software subsystems – the development cycles were misaligned.

SKA is fairly positive: 77% rated it from non-existent to only moderate, as seen in Fig. 16. From the 20 responses given to the open-ended Question 4.7, most of them were categorized as SAFe Related (g=10) (see Table A.8), and provided further insights about the practices related to this perception. The responses coded with said category pointed out that hardware and software development has been aligned through SAFe's Program Increments (PIs). More specifically, it was mentioned that this has been working successfully in one of the teams[10] by pulling the firmware/FPGA teams into SAFe, making these components formally part of the PI's development process (in other words, mirroring these components development with the SAFe agile). Interestingly, the practice of using automated artifacts generation workflows, supported by machine-readable interface specifications (Machine Readable Formalisms (g=4) discussed in Section 4.4), was also mentioned in this question by members of the same ART's team (three out of the five that participated in the study). This arguably means that this is a local practice with a seemingly positive impact on aligning the two life-cycles, and as a mean to properly capture the mutual interfacing expectations of the involved hardware/firmware/software parties.

5. Answers to research questions

In the following section, we discuss the results of the study by answering each of the research questions identified in Section 3.

RQ1.1: To what extent the harmonization practices identified in LOFAR+, as applied in the context of the SKA, address the same kind of issues on requirements?

For systems of the scale of SKA, **a formal SE process where the requirements were defined up-front *(B1)*, has been key not only in establishing properly defined interfaces *(P6)* and requirements *(P1)*, but also in preventing essential features from being missed down-the-line of the development process** (e.g., in lower level requirements and within the agile process). According to the survey results (see Section 4.2), this seems to be the case especially when working in combination with an agile process, where the understanding of the system is expected to emerge over time. However, **the results also suggest a friction between these two engineering disciplines (*traditional systems engineering* and *scaled agile*), which seems to be related to the said front-loaded requirements**: on the one hand, it is perceived that working with an agile approach could lead to lose the focus on the original requirements (e.g., through working by assumptions rather than requirements); on the other hand, it is perceived that the provided system-level requirements, by being long-term ones, are difficult to follow within the short-term phases of an agile process (see Section 4.2.1). This could explain, to some extent, why some of the consequences of having incomplete requirements identified in LOFAR+ are seemingly present also in the SKA, namely the need for making assumptions while working with them, and the little influence such requirements have on early test design.

RQ1.2: To what extent the harmonization practices identified in LOFAR+, as applied in the context of the SKA, address the same kind of issues on subsystem design decisions?

There is a general perception that, given the current setting of the SKA project, in most cases local design decisions are not having a negative impact on the overall system design and qualities (see

[10] CSP Interface Prototyping Agile (CIPA) Team.

Section 4.3). Both the early partition of the system *(B3)* and the hierarchical co-architecting *(B2)* seemingly contributed to avoid such a negative impact *(P4)*. It is therefore fair to conclude that **both practices (*(B2)* and *(B3)*) can be seen as confirmed, albeit to a different extent.** Although practice *(B3)* has mostly a moderate to positive perception in this regard, it is seen by others as something that hindered the Solution Intent, as it led to an over-complicated architecture. However, this is arguably due to such an early partition being guided by a consortium division rather than by logical products. In contrast, practice *(B2)*, not only has a more positive perception, but also emerged (on the open-ended responses) as a key practice to address the impact of changes at the subsystem-level *(P5)*. However, the survey results also revealed particular instances where, despite practices *(B3)* and specially *(B2)* being applied, local design decisions had (to a minor extent) a negative impact on the system qualities. As such instances are related to local decisions guided by the misinterpretation of the requirements, or a personal experience bias, this could be another call for the introduction of intermediate, short-term requirements as discussed for *RQ1.1*.

At the same time, further analysis of the survey results also revealed a new category of local design decisions that had a negative impact on the system architecture and qualities: *workarounds to deal with higher-level, prescriptive software design decisions*. That is to say, early prescribed design decisions appear to have constrained the space for making decisions at the subsystem level, which led to workarounds to fulfill the subsystem goals. This, in turn, led to (sub-)system inefficiencies. Interestingly, the discussion of these two kinds of 'negative' local design decisions (the ones related to requirements, misinterpretations or personal bias, and the ones forced by higher-level decisions), shows an apparent contradiction between: (a) the participants that call for a more holistic view of the system early on for making local design decisions; and (b) the participants that call for less prescriptive high-level decisions. This is seemingly another sign of friction between the two engineering perspectives combined in the SKA. Further investigation on the causes of this friction is required.

> *RQ1.3: To what extent the harmonization practices identified in LOFAR+, as applied in the context of the SKA, address the same kind of issues on subsystem interfacing?*

Practices for early integration such as *harware/firmware tryouts* and *hardware simulations (B4)* **appear to be confirmed as an approach to create interfaces that fulfill the mutual expectations of their involved parties** *(P7)* (see Section 4.4). Furthermore, it is worth noting that these tend to be used **in conjunction with formal ICDs and other alternative specifications** (discussed in Section 5) for this purpose. This is arguably related to the fact that, according to the results, there are particular subsystems where high-level interfaces, or ICDs, are too vague and incomplete (although this is seen as a deliberate decision by some). Among the elements referred as commonly missed on the ICDs were the traceability of the requirements, and the time- and state- behavioral details, which was also a recurrent theme in the LOFAR+ case. Furthermore, it is worth highlighting that the early partition of the system *(B3)* emerged when exploring this research question with a negative connotation: it was perceived that this early partition, by leading to an overcomplicated architecture discussed in Section 5, contributed to the emergence of obscure or not properly documented ICDs.

> *RQ2—What additional cross-disciplinary harmonization practices have been adopted by the SKA?*

In the following, we present a summary of additional practices for harmonizing architecting processes in Systems Engineering and Software Engineering, as identified by the practitioners, and their relation with the cross-disciplinary issues already identified.

(B5) Scaled agile in the context of a traditional systems engineering process.

As discussed in RQ1.1, requirements defined up-front through a rigorous SE process are seen as key in large-scale systems like the SKA. Therefore it is not surprising, as described in the LOFAR+ case study, that unclear or incomplete requirements – which happens to a lesser extent also in the SKA – are among the causes of major integration or operational issues. The results of the data analysis (see Section 4.2.3) show that **the adoption of a scaled agile framework like SAFe is seen as the key element to address unclear/incomplete requirements *(P1)***, as agility has shown to be successful when it comes to filling the gaps as the system evolves. Furthermore (as discussed in Section 4.5), **the *Program Increments* of the adopted agile framework, when including the firmware/FPGAs teams, is seen as a successful approach for aligning hardware and software development cycles**, an issue that was un-addressed in the context of LOFAR+ *(P8)*.

(B6) Cross-level communication and transparency

In regard to practices that prevent the negative impact of local design decisions in the overall system *(P4)*, the results suggest that **above any rigorous process, transparency and communication across different levels within the project, are key to prevent and mitigate this kind of issues** (see Section 4.3). This includes not only giving access to the right people across the organization, but also adopting strategies like the *Communities of Practice* (CoP), which provides a loose structure that brings together people with common concerns, and enables cooperation across multiple domains [30]. Along the same lines, communication between the architects involved at both ends of the ICDs seems to be essential in order to achieve consensus-driven solutions. Overall, it is worth noting that the perception that *(P4)* is an issue that happens to a minor extent on the SKA, suggests the positive influence this communication-oriented practice has had for its mitigation.

(B7) Intermediate interfacing specifications and automation workflows

According to the results (see Section 4.5), **the issues of lack of alignment between hardware and software development cycles *(P8)* and of mutual expectations not captured by interfaces *(P7)*, are being successfully addressed – by some groups within SKA – by means of a Model Driven-like approach.** In this approach, the low-level details of the interfaces between hardware and software (which are originally defined in human-only-readable documents, e.g., ICDs) are captured and managed as the source of truth for both parties, through a machine-readable formalism. This formalism, in turn, has allowed the development of tools and workflows for the automatic generation of key artifacts for both hardware/software parties as the resulting subsystem evolves. However, as this practice has been evidenced only in a few groups within the SKA, it is worth to be further explored and compared with its counterpart in the microcontrollers industry, from which it was inspired. In contrast, regarding issues yet to be addressed, **it is worth noting that the lack of time-behavioral details on both requirements and interfacing specifications *(P9)* seem to be a recurrent theme in (ultra-)large-scale systems like LOFAR+ and the SKA**.

6. Implications for practitioners and researchers

The following summarizes our understanding of the implications of this study's results for practitioners and researchers.

6.1. Agile and Systems Engineering principles in the context of software-intensive SoS

The plan-driven methods prescribed by the traditional SE principles have been historically shown to provide a logical process structure for large-scale, multidisciplinary SoS projects [31]. However, when the projects are software-intensive, such plan-driven approaches have

been often criticized, as they make the development of said software elements difficult [32]. This is particularly due to the assumption that changes in software, like in hardware and mechanical systems, will decrease over time [27]. For this reason, the relationship between the Agile and Systems Engineering principles has been extensively explored, leading to multiple approaches for mixing them, such as the ones supported by *Model-based Systems Engineering*, where agile principles are applied on top of evolving, high-precision system models [33, 34].

The SKA, on the other hand, and arguably due to its scale, has followed a tailored approach that combines both sets of principles by following a traditional plan-driven design, and then, based on the defined requirements and interfaces, transitioning to a scaled agile process. The results of our study highlight the importance of system requirements properly defined up-front in this particular and similar approaches, as they capture key features that cannot be missed down the line in the agile process, and that will not necessarily arise as part of the emergent understanding of the system. However, these requirements seem to be an important factor of friction between the SE perspective of a system, and the one from the agile teams working on the software-heavy elements of it, due to its long-term nature. For this reason, practitioners in this context should take into consideration the importance of (1) **deriving intermediate (short-term) requirements from the long-term ones, so that agile teams are less prone to lose the focus on them**, and (2) **ensure and continuously validate the traceability between long-term and short-term requirements**.

6.2. System decomposition in large-scale, directed SoS

Decomposing complex, large-scale systems (of systems) into subsystems through e.g. logical decomposition is a core SE practice, which, with the use of formal interfacing specifications (e.g., ICDs) defines clear boundaries that allow multiple teams to work concurrently on the system design and development. In the LOFAR+ case, the transition from a layer-based decomposition (hardware/firmware/software) to a subsystem-based one, showed that this SE practice helped in preventing local design decisions from having a negative impact on the overall architecture and system qualities. Likewise, on the SKA project, this practice was key in delegating the design of such a massive system to multiple consortia. However, as this decomposition was guided by delegation of responsibilities rather than logical products, which had adverse implications on the system qualities; this suggests the need for further **research on reconciling functional decomposition with organizational decomposition at this scale**. Furthermore, the communication-oriented practices adopted by the SKA regarding ICDs, that is to say, the work towards consensus-driven solutions with the architects involved at both ends of the ICDs, calls for further **research on how to achieve this communication effectively, given the potentially very different knowledge domains involved**.

6.3. Driving and guarding the system qualities

In the LOFAR+ case study multiple scenarios were identified where subsystem-level design decisions (e.g. when boundaries between subsystems were not clearly demarcated) had a negative impact on the overall system architecture and its qualities. In the SKA, seemingly due to the co-architecting roles and the overall scaled agile process already in place, this study revealed other kind of design decisions that, indirectly, also have a negative impact; admittedly, this happens to a minor extent. High-level, prescriptive, software-related technical decisions (e.g., algorithms parameters, protocols selection, abstraction layers, etc.) lead practitioners to resort to workarounds on the subsystems implementation to fulfill their goals. These workarounds however end up hindering the overall system qualities. This calls practitioner to give **higher priority to the negotiation of decisions with a cross-cutting impact on multiple agile teams in the context of a scaled agile framework**. Furthermore, the apparent contradiction between the engineers that call for a more holistic view of the system to make better informed decisions, and those that call for less high-level prescriptive specifications for the system calls for further **research on practices for reconciling the bottom-up and top-down design approaches that coexist in systems of this scale**.

6.4. Subsystems interfacing

This study corroborated the importance of formal interface specifications as enablers of effective collaboration between cross-disciplinary teams [35]. However, it also confirmed that the lack of details about the dynamic aspects of the interfaces (e.g., time and state-behavioral details) is a recurring phenomenon that is seemingly causing problems (mostly) on the software side of the involved subsystems. This suggests the need for further **research on formalisms or Domain Specific Languages (DSLs) for the specification of such elements**, especially considering the cross-disciplinary settings of large-scale systems like LOFAR+ and the SKA and the differences in the terminology they exhibit, as discussed by previous studies in SE-SWE interfacing [36].

At the same time, the study further revealed what seems to be a promising best practice – given the perceived success within the team that adopted it – for aligning the life cycles of hardware and software teams, as well as managing the lower-level interfaces of such teams: the use of machine-readable interfacing specifications as a single source of truth for both hardware and software teams, and workflows for the generation of relevant artifacts for both parties. Although this practice is well known in the semi-conductors domain, its success in the context of large-scale systems calls for further **research on how to make it consistent with the higher-level interfaces (i.e., ICDs) that are often defined in human-only readable formats**, and from which the lower-level interfaces are actually meant to be derived.

7. Threats to validity

In the following we discuss potential threats to the validity of this study, and the steps taken to mitigate them. We use Runeson and Höst [25] as a guide for this purpose. *Internal validity* does not apply as this study does not examine causalities [37].

Construct validity. Construct validity refers to the degree the research instruments, in this case the online survey, are consistent with the research questions, and reflect what the researchers have in mind while designing them. To improve the construct validity of the study, as discussed in Section 3, the survey was piloted by three engineers with experience in the case subjects of this and the previous study (the SKA and LOFAR+ projects). They provided feedback about the wording and the terminology used in the questions. The survey was also reviewed prior to its promotion to the target population, by one of the co-authors, who also has experience in the application domain, as well as one of the SKA architects.

To prevent biasing the perception of the participants about the results of the previous study, we designed the survey without any presumption of expectation of familiarity with the LOFAR+ case. To do so, as described in Section 3, we first explored how such practices have been applied in the SKA, so that the survey questions can be designed using SKA-specific context and terminology. Furthermore, and to avoid bias by leading questions and make the confirmation more reliable, the survey did not use close-ended questions to directly gauge the relationship between specific practices and issues (e.g., *to what extent practice A improves issue X?*). Instead, such relations were mostly evaluated through the themes (or codes) that emerged from the open-ended responses to questions of the type *how has the SKA prevented or mitigated X?*, and complemented by the insights given by the closed-ended ones. Independent verification of this process is possible through the replication package of this study.

Another construct-related issue that can affect negatively the confirmatory robustness of our study is due to the risk of the number of opted-out questions per role (described in Section 4.1) skewing the data. Likewise, this threat was mitigated, to some extent, by accompanying them with related open-ended questions. Ultimately, we believe that the risk due to opt-outs is justified when compared to the ones created by forcing participants to provide an answer to a question that they are not comfortable or familiar with. Doing so would either introduce noise in the replies due to participants choosing the middle/neutral option as the default response, or create mid-survey drop-out issues with frustrated participants.

External validity. External validity refers to the extent research findings can be generalized from the sample to the target population. In this case, external validity can be seen from two viewpoints: (1) the SKA project as a sample of all the directed SoS, and (2) the study participants as a sample of total population of the engineers involved on the SKA's ART. With respect to the former, although this study is conducted in the same application domain (radio-astronomy scientific instruments) as the one it aims to confirm and extend, we believe that its characteristics make it still representative of large-scale, directed SoS. However, validation of the findings in other engineering domains remains part of our future work. Regarding the latter, this study was subject to self-selection bias due to its non-probabilistic sampling—the participation to the study was based on an open call to the target population. However, as described in Section 4.1, we find the sample representative enough not only of the roles (considering that all of them are involved on the ART), but also on their affiliations and distribution across the globe. The number of participants was relatively limited (46), but this was due to the very specific target population of the engineers involved in the SAFe process, which was focused to the Software-dominant components of the project.

Reliability. Reliability refers to the extent that the data and its analysis are dependent on the researchers that conducted the study. To mitigate researcher bias in the process, two of the authors were involved in the quantitative data analysis, and came to an agreement with a third author about its interpretation given the results of the qualitative data analysis. The qualitative data analysis, in turn, was conducted by the first author, and its outcome validated by other two authors. Furthermore, both qualitative and quantitative analysis can be verified through the replication package provided for this purpose.

8. Conclusions

The concept of System of Systems has gained significant attention from multiple engineering disciplines, as a powerful concept to deal with the complexity of modern systems and the goals they aim to attain. This is particularly the case for Systems Engineering (SE) and Software Engineering (SWE), two disciplines that are deeply intertwined in the development of many engineered[11] SoS: the former managing the system complexity and the disciplines involved in it (including SWE), and the latter providing key features through the software-heavy components scattered across it. Despite this mutual dependency, however, there is little research on how to harmonize these disciplines when working together, in particular regarding their respective architecting processes. This study – the third one in a series of related studies – contributed to closing this gap by exploring the architecting harmonization practices previously identified in the LOFAR+ case when applied in the context of the SKA. Based on its results, it evidently confirms the findings of the LOFAR+ study with respect to such practices. Furthermore, it unveiled additional best practices and provided further insights to be considered by practitioners when architecting SoS, and to be explored by researchers in the area.

[11] In contrast to naturally-occurring ones.

In particular, this study shows that an agile process is indeed a good complement for traditional SE processes when it comes to incomplete/vague system requirements, something arguably unavoidable in systems with the scale of the SKA. Likewise, strategies that enable discussion on common concerns across the systems, and custom interfacing management approaches, are further examples of practices that complement the SE practices (e.g., the *critical design reviews*) in preventing the negative impact of local design decisions, and aligning development cycles, respectively. The areas of improvement identified in this case study, on the other hand, provide important insights to be considered in future projects and in the aforementioned research agenda. For instance, this study suggests that having short-term intermediate requirements (derived from the long-term ones) would improve the alignment between the system originally envisioned on the early stages of the project (when such requirements are defined), and the system understanding that emerges from the agile process. When it comes to the early partition of the system, and their demarcation with ICDs, this study highlights the importance of further attention – and further research – on (1) system breakdown techniques that better align functional decomposition with large-scale responsibilities distribution, and (2) the way software-specific design decisions, with impact across multiple agile teams are made, as a measure to prevent architectural smells.

Overall, using the SKA has exemplified how, with the right practices rather than a 'clash of cultures', disciplines can complement each other while combining the best of both worlds. This makes us believe that this work can be seen as the baseline for a new direction in the current SWE research agenda in SoS architecting: *well-informed practicing of SWE in the context of SoS projects governed by a SE process*.

Along this line, as a future work we plan to build on the findings and work towards a contribution to some of the prevalent issues identified by this and previous studies, and particularly the communication-related ones, concerning terminology, completeness, and consistency. For this purpose, we plan to devise processes and artifacts for the management of cross-disciplinary interfaces that work under some of the aforementioned practice-related principles, namely transparency and consensus-oriented communication, in combination with the automation possibilities of modern technical documentation, whose potential has been shown in this and other domains in the literature.

CRediT authorship contribution statement

Héctor Cadavid: Conceptualization, Methodology, Investigation, Writing – original draft, Formal analysis, Data curation. **Vasilios Andrikopoulos:** Supervision, Conceptualization, Methodology, Formal analysis, Writing – review & editing. **Paris Avgeriou:** Supervision, Methodology, Writing – review & editing. **P. Chris Broekema:** Conceptualization, Resources, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

A replication package is provided (link to the digital academic repository included in the paper).

Acknowledgments

The authors would like to thank the members of the SKA organization and the engineers involved in the project for their time in helping us conduct this research. Special thanks to Nick Rees, Juande Santander-Vela and Marco Bartolini for their support and feedback while conducting and promoting this study within the 10th PI-planning event. We would also like to particularly thank the participants of the survey pilot for their invaluable input: Marcel Loose, Stefan Wijnholds, and Ágnes Mika.

References

[1] DoD Architecture Framework Working Group and others, DoD architecture framework version 1.0, Department of Defense, 2003.

[2] Arianna Alfieri, Marco Cantamessa, Francesca Montagna, The SoS approach for lean manufacturing systems, Int. J. Technol. Manage. 57 (1/2/3) (2012) 149–165.

[3] Ursula Rauschecker, Simon J. Ford, Nikoletta Athanssopoulou, Developing a vision for multi-site manufacturing system of systems, in: Enabling Manufacturing Competitiveness and Economic Sustainability, Springer, 2014, pp. 79–84.

[4] Duncan Ki-Aries, Shamal Faily, Huseyin Dogan, Christopher Williams, System of systems characterisation assisting security risk assessment, 2018.

[5] Ali Mostafavi Zachry, Chao Fan, Establishing a framework for disaster management system-of-systems, in: Systems Conference (SysCon), 2018 Annual IEEE International, IEEE, 2018, pp. 1–7.

[6] Jan-Philipp Steghöfer, Gerrit Anders, Florian Siefert, Wolfgang Reif, A system of systems approach to the evolutionary transformation of power management systems., in: GI-Jahrestagung, 2013, pp. 1500–1515.

[7] Amit J. Lopes, R. Lezama, Ricardo Pineda, Model based systems engineering for smart grids as systems of systems, Procedia Comput. Sci. 6 (2011) 441–450.

[8] Alex Gorod, Susan Merchant, Leonie Hallo, Toward systemic governance of cancer treatment as a system of systems, in: 2018 13th Annual Conference on System of Systems Engineering (SoSE), IEEE, 2018, pp. 556–560.

[9] Suguru Okami, Naohiko Kohtake, Transitional complexity of health information system of systems: Managing by the engineering systems multiple-domain modeling approach, IEEE Syst. J. (2017).

[10] Draft BS ISO/IEC 21839 Information technology - Systems and software engineering - System of Systems (SoS) considerations in life cycle stages of a system, Standard, International Organization for Standardization, Geneva, CH, 2018.

[11] Mark W. Maier, Architecting principles for systems-of-systems, Systems Engineering: The Journal of the International Council on Systems Engineering 1 (4) (1998) 267–284.

[12] J. Boardman, S. Pallas, B.J. Sauser, et al., Report on system of sys-tems engineering, final report for the office of secretary of defense, Technical report, Stevens Institute of Technology, Hoboken, NJ, 2006.

[13] Andrea Ceccarelli, Andrea Bondavalli, Bernhard Froemel, Oliver Hoeftberger, Hermann Kopetz, Basic concepts on systems of systems, in: Cyber-Physical Systems of Systems, in: Lecture Notes in Computer Science, Springer, Cham, 2016, pp. 1–39.

[14] Michael Gagliardi, W.G. Wood, J. Klein, J. Morley, A uniform approach for system of systems architecture evaluation, CrossTalk 22 (3–4) (2009) 12–15.

[15] Héctor Cadavid, Vasilios Andrikopoulos, Paris Avgeriou, Architecting systems of systems: A tertiary study, Inf. Softw. Technol. 118 (2020) 106202.

[16] Gerrit Muller, Validation of systems engineering methods and techniques in industry, Procedia Comput. Sci. 8 (2012) 321–326.

[17] Héctor Cadavid, Vasilios Andrikopoulos, Paris Avgeriou, John Klein, A survey on the interplay between software engineering and systems engineering during SoS architecting, in: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), in: ESEM '20, Association for Computing Machinery, 2020, pp. 1–11.

[18] Héctor Cadavid, Vasilios Andrikopoulos, Paris Avgeriou, P. Chris Broekema, System- and software-level architecting harmonization practices for systems-of-systems – an exploratory case study on a long-running large-scale scientific instrument, in: 2021 IEEE International Conference on Software Architecture (ICSA), IEEE, 2021, pp. 13–24.

[19] Judith S. Dahmann, Kristen J. Baldwin, Understanding the current state of US defense systems of systems and the implications for systems engineering, in: 2008 2nd Annual IEEE Systems Conference, IEEE, 2008, pp. 1–7.

[20] Michael P van Haarlem, Michael W Wise, AW Gunst, George Heald, John P McKean, Jason WT Hessels, A Ger de Bruyn, Ronald Nijboer, John Swinbank, Richard Fallows, et al., LOFAR: The low-frequency array, Astron. Astrophys. 556 (2013) A2.

[21] Jason Hessels, KNAW-agenda grootschalige onderzoeksfaciliteiten, 2016, (accessed December 6, 2019).

[22] Kim Fowler, Developing and Managing Embedded Systems and Products: Methods, Techniques, Tools, Processes, and Teamwork, Elsevier, 2014.

[23] Dean Leffingwell, SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises, Addison-Wesley Professional, 2018.

[24] Victor R. Basili, Software modeling and measurement: the Goal/Question/Metric paradigm, Technical report, 1992.

[25] Per Runeson, Martin Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2) (2009) 131.

[26] Satu Elo, Helvi Kyngäs, The qualitative content analysis process, J. Adv. Nursing 62 (1) (2008) 107–115.

[27] Juande Santander-Vela, Agile software engineering and systems engineering at SKA scale, 2017, arXiv preprint arXiv:1712.00061.

[28] Susanne Friese, Qualitative Data Analysis with ATLAS. Ti, Sage, 2019.

[29] Richard E. Fairley, Systems Engineering of Software-Enabled Systems, John Wiley & Sons, 2019.

[30] Richard Knaster, Dean Leffingwell, SAFe 4.5 Distilled: Applying the Scaled Agile Framework for Lean Enterprises, Addison-Wesley Professional, 2018.

[31] Reinhard Haberfellner, Peter Nagel, Mario Becker, Alfred Büchel, Heinrich von Massow, Systems Engineering, Springer, 2019.

[32] Ronald S. Carson, 4.2. 1 can systems engineering be agile? Development lifecycles for systems, hardware, and software, in: INCOSE International Symposium, Vol. 23, Wiley Online Library, 2013, pp. 16–28.

[33] Sally Muscarella, Macaulay Osaisai, Sarah Sheard, Systems and software interface survey, in: INCOSE International Symposium, Vol. 30, Wiley Online Library, 2020, pp. 1294–1312.

[34] Bruce Powel Douglass, Agile Systems Engineering, Morgan Kaufmann, 2015.

[35] Sally Muscarella, Macaulay Osaisai, Sarah Sheard, Systems and software interface survey, in: INCOSE International Symposium, Vol. 30, Wiley Online Library, 2020, pp. 1294–1312.

[36] Sarah Sheard, Michael E. Pafford, Mike Phillips, Systems engineering–software engineering interface for cyber-physical systems, in: INCOSE International Symposium, Vol. 29, Wiley Online Library, 2019, pp. 249–268.

[37] Robert K. Yin, Case Study Research and Applications: Design and Methods, Sage publications, 2017.