A Survey on the Interplay between Software Engineering and Systems Engineering during SoS Architecting

Cadavid Rengifo, Hector; Andrikopoulos, Vasilios; Avgeriou, Paris; Klein, John

Published in:
Proceedings of the International Conference on Empirical Software Engineering and Measurement, 2020

DOI:
10.1145/3382494.3410671

Publication date:
2020

A Survey on the Interplay between Software Engineering and Systems Engineering during SoS Architecting

Héctor Cadavid*
h.f.cadavid.rengifo@rug.nl
University of Groningen
Groningen, the Netherlands

Vasilios Andrikopoulos
v.andrikopoulos@rug.nl
University of Groningen
Groningen, the Netherlands

Paris Avgeriou
p.avgeriou@rug.nl
University of Groningen
Groningen, the Netherlands

John Klein
john.klein@computer.org
Gloucester, Massachusetts

ABSTRACT

Background: The Systems Engineering and Software Engineering disciplines are highly intertwined in most modern Systems of Systems (SoS), and particularly so in industries such as defense, transportation, energy and health care. However, the combination of these disciplines during the architecting of SoS seems to be especially challenging; the literature suggests that major integration and operational issues are often linked to ambiguities and gaps between system-level and software-level architectures.

Aims: The objective of this paper is to empirically investigate: 1) the state of practice on the interplay between these two disciplines in the architecting process of systems with SoS characteristics; 2) the problems perceived due to this interplay during said architecting process; and 3) the problems arising due to the particular characteristics of SoS systems.

Method: We conducted a questionnaire-based online survey among practitioners from industries in the aforementioned domains, having a background on Systems Engineering, Software Engineering or both, and experience in the architecting of systems with SoS characteristics. The survey combined multiple-choice and open-ended questions, and the data collected from the 60 respondents were analyzed using quantitative and qualitative methods.

Results: We found that although in most cases the software architecting process is governed by system-level requirements, the way requirements were specified by systems engineers, and the lack of domain-knowledge of software engineers, often lead to misinterpretations at software level. Furthermore, we found that unclear and/or incomplete specifications could be a common cause of technical debt in SoS projects, which is caused, in part, by insufficient interface definitions. It also appears that while the SoS concept has been adopted by some practitioners in the field, the same is not true about the existing and growing body of knowledge on the subject in Software Engineering resulting in recurring problems with system integration. Finally, while not directly related to the interplay of the two disciplines, the survey also indicates that low-level hardware components, despite being identified as the root cause of undesired emergent behavior, are often not considered when modeling or simulating the system.

Conclusions: The survey indicates the need for tighter collaboration between the two disciplines, structured around concrete guidelines and practices for reconciling their differences. A number of open issues identified by this study require further investigation.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; *Ultra-large-scale systems.*

KEYWORDS

systems of systems, architecting, practitioners survey

ACM Reference Format:
Héctor Cadavid, Vasilios Andrikopoulos, Paris Avgeriou, and John Klein. 2020. A Survey on the Interplay between Software Engineering and Systems Engineering during SoS Architecting. In *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3382494.3410671

1 INTRODUCTION

The concept of System of Systems (SoS) is used across application domains like defense, automotive, energy and health care to describe a family of systems that cooperate to provide new capabilities [24]. Strictly speaking, most systems could be considered as SoS, since they can be decomposed into smaller subsystems [19]; however, the *engineered SoS* — in contrast to a naturally occurring one like an ecosystem — is often distinguished by some degree of *operational and managerial independence* of its constituents, as proposed by Maier's seminal paper [28]. The SoS paradigm is of strategic importance in industry, as it promotes the cooperation among already operational systems towards the delivery of new capabilities [5].

The architecting process of an SoS, compared with other engineered systems, must address a unique set of challenges. For

*Also with Escuela Colombiana de Ingeniería.

ESEM '20, October 8–9, 2020, Bari, Italy

ACM ISBN 978-1-4503-7580-1/20/10…$15.00
https://doi.org/10.1145/3382494.3410671

instance, architecting an SoS involves the adaptation and integration of existing and usually heterogeneous independent systems as constituent systems of the SoS. Such integration should enable the fulfillment of the SoS mission or capabilities through cooperation between its constituent systems [7]. However, those constituent systems are fundamentally independent, and their own needs must also be balanced with the needs of the SoS that they are integrated into [34]. At the same time, the architecting process of most modern engineered SoS must follow the complementary perspectives of two related but distinct disciplines: Systems Engineering (SE) and Software Engineering (SWE). Historically, the development of SoS has usually been driven by SE processes [33] due to the integration of physical (e.g. mechanical, electronic) constituent systems. However, software is now not only pervasive and abundant, but also a critical element of the performance and features offered by most modern engineered systems [17, 35].

Unfortunately, the interplay between the two disciplines has proven problematic. Integration and operational problems in SoS are often linked to inconsistencies or gaps between the system-level and the software-level architectural elements created, respectively, by these two disciplines [12, 20]. This seems to be a consequence of the way SE and SWE have evolved over time: starting with common roots but becoming misaligned due to their separate evolutionary paths [35]. The differences in their approaches make the architecture perspectives difficult to synchronize, and as Sheard et al. [40] reported, these differences can interfere with each discipline's practices. For example, in a project using a SE decomposition approach, the design and development of the software components of the system may become distributed across several physical sub-systems and assigned to independent development teams or contractors. In that case, the architectural concerns of the system (e.g. performance, reliability, security) would be difficult to address in a unified way at the software level.

The need to reconcile the SE and SWE disciplines has driven a number of initiatives to integrate them at a development process level. For instance, the ISO 15288 standard for SE processes [23] includes suggestions for its integration with ISO 12207-compliant [41] SWE processes. However, most methodologies and guidelines tailored for SoS are mostly SE-centered and lack guidelines for the integration of SWE architecting practices. Prominent examples of such methodologies and guidelines are the *Systems Engineering Guide for SoS* [34], the ISO/IEC 21839 standard [24], and the results of the *FP7*[1] projects DANSE [2], COMPASS [1] and AMADEOS [3]. Nevertheless, none of these approaches prescribes practices to align software-level with system-level architectures, as they consider software as just another type of constituent system and not as a cross-cutting element of an SoS.

Therefore, to contribute to the improvement of the architecting processes of engineered SoS, we believe it is important to address the problems posed by the interplay between SWE and the SE disciplines. There is little up-to-date evidence about these problems in 'traditional' engineered systems, and to the best of our knowledge, no evidence at all for the particular case of SoS. This work addresses this gap by identifying, from the perspective of practitioners from both disciplines, *the challenges that arise from the interplay of these two disciplines when architecting systems with SoS characteristics*. To this end, this paper reports on the design, execution and main findings of an online survey of practitioners working on complex systems with SoS characteristics that took place between December 2019 and February 2020.

The rest of this paper is structured as follows: Section 2 summarizes the state of the art on the interplay between SE and SWE in and out of the context of SoS systems. Section 3 presents the design of the practitioner survey, and Section 4 summarizes its results. Section 5 discusses the relevance of our findings for practitioners and researchers, and outlines open issues for further research. Finally, Section 6 concludes this study.

2 RELATED WORK

The interplay between SE and SWE in the context of SoS architecting has not been widely studied. We found only two studies exploring this issue, each focused on a particular scenario. Boehm et al. [8] focused on the acquisition and integration of constituent systems, and proposed an approach to address the problem of underperforming SE in the domain of defense caused by the traditional hardware-centered systems engineering and subsystem acquisition practices. Gagliardi et al. [20] proposed an evaluation method for SoS and software architectures to address the problem of lack of attention of quality attributes caused by the diversity of notations used for system and software elements of SoS.

Beyond the context of architecting in the specific domain of SoS, there is related work exploring the problems between these two disciplines in general and specific interdisciplinary problems in the architecting process of systems. Here, we provide a brief description of both categories. In one of the earliest works in the first category, Sommerville [42] pointed out how failed case studies like the DIA Baggage System [15] were caused not only by software problems, but by how software was integrated in the systems engineering process. He proposed the introduction of systems engineering concepts in computer science courses as an approach to tackle this integration problem. Much later, Fairley et al. [18] discussed how software engineers' lack of qualifications on SE concepts limited their participation in system-level decision making. Like Sommerville, they proposed to address this problem at its roots by integrating SWE topics in SE curricula. This, and Fairleys's follow-up works on alternative development models for SE and SWE process articulation, led to the recently published book *Systems-Engineering for Software-enabled systems* [17].

In the second category of related work, Maier's paper on System and Software Architecture reconciliation [29] is one of the earliest studies that takes an architectural perspective on the SE-SWE interrelation problem. In this work, which later would become part of *The art of systems architecting* book [30], Maier pointed out the problems caused by mismatches between the architectural structures used by the traditional, hardware-centered systems engineering and those used by modern software engineering.

In recent years, INCOSE[2] has promoted an empirical approach to tackle the problems between these two disciplines. For instance, Pyster et al. [35] reported on a workshop co-sponsored by INCOSE

[1] Seventh framework programme of the European Commission for research and technological development including demonstration activities

[2] International Council on Systems Engineering

with 29 professionals from academia and industry described as a "cross section of systems and software engineering community", held to identify said problems, and the challenges created by them. Subsequently, INCOSE approved the charter of the *Systems and Software Interface Working group* (SaSIWG), and recently published a series of problem-related findings identified through its members interaction [39, 40].

3 STUDY DESIGN

3.1 Study goal and research questions

The goal of this study is to identify the problems related to the interplay between SE and SWE disciplines during SoS architecting in practice. We used the five parameters proposed by the Goal-Question-Metric (GQM) goal template [6] for a precise definition:

> **Analyze** *the architecting process of complex engineered systems with SoS characteristics* **for the purpose** *of collecting and characterizing the perceived challenges* **with respect to** *the interplay between the Systems and Software Engineering disciplines* **from the viewpoint of** *experts with practical experience and background in Systems Engineering, Software Engineering or both*, **in the context of** *application domains where said interplay takes place such as defense, energy, transportation and health.*

The first parameter — the *Object* under study, focuses on systems with *SoS characteristics*, rather than systems *self-identified* as SoS; this is because of the unclear boundaries of SoS as a concept. For instance, not all systems that present SoS characteristics self-identify with this label [26]; some adopt a related term (e.g. Cyber-Physical Systems or Software-intensive Systems [10]). This could be due to the debate regarding whether an SoS is a distinct class of systems [32], or whether it is an optional viewpoint for complex systems [14]. On the other hand, there are many cases of mis-classifying systems as SoS, as noted by Maier [28]. This problem is suggested, for example, by the way SoS as a concept is used in some secondary studies in the field [10], with no references to any of the multiple existing definitions or sets of characteristics [7, 19]. As a way to deal with these semantic ambiguities, we instead adopt the *SoS characteristics* from Firesmith's model [19] to classify the systems used as reference for this study (see Section 3.2.1):

Constituent System Autonomy The degree to which the constituents are operationally independent, i.e. with a purpose of their own.

Constituent System Governance The degree to which the constituents are managed, owned or operated by a higher-level authority.

Constituent System Heterogeneity The degree to which the constituents differ from each other in terms of functionality or architecture.

Constituent System Physical distribution The degree to which the constituents exist in geographically disperse locations.

SoS Complexity The degree to which the SoS is difficult to understand and analyze.

SoS Evolution The degree to which the SoS requirements change over time.

SoS Emergence The degree to which desired or undesired behaviors emerge from the cooperation between the constituent systems.

Based on the goal described above, we defined the following research questions:

RQ1 What challenges do practitioners face as a consequence of the interplay between the Software Engineering and Systems Engineering disciplines during SoS architecture analysis, synthesis and evaluation?

RQ2 What challenges do practitioners face in SoS architecting when dealing with the emergent behavior of SoS, and the autonomy of their constituent systems (ConS)?

The first research question (*RQ1*) seeks to identify the problems faced by practitioners due to the mismatches between the SE and SWE disciplines in the architecting process of an SoS. Specifically, we adopt the reference architecting process of Hofmeister et al. [22], which considers three activities: (1) *analysis*—dealing with system requirements, (2) *synthesis*—the way architectural decisions are taken, and (3) *evaluation* of such architectural decisions. We consider each of these three activities from the standpoint of both systems and software engineering. The second research question (*RQ2*), aims to identify relevant challenges in the architecting process explicitly linked to two of the SoS characteristics: emergent behavior and autonomy of the constituents of the system. We have selected these two as they are the most distinctive characteristics of SoS [21, 28].

3.2 Research method

To answer the research questions, we conducted a questionnaire-based survey to collect insights from practitioners with experience in the architecting of systems with SoS characteristics. The survey research method was selected because the research questions require a broad overview of the studied object (*architecting of systems with SoS characteristics*) and examination of knowledge, attitudes and behaviors related to it (*interdisciplinary problems experienced by practitioners*) [25]. More specifically, we use the online survey method to obtain information from as wide a sample as possible from the population of practitioners and researchers with practical experience. We adopted the guidelines of Kitchenham and Pfleeger [25] and followed the prescribed steps:

(1) Setting the objectives, as described in Section 3.1.
(2) Designing the survey.
(3) Developing the survey instrument.
(4) Obtaining valid data.
(5) Analyzing the data.

3.2.1 Survey design and development. The questionnaire designed for this study has 24 questions, which include both multiple-choice and open-ended questions, divided into three sets. The first set focuses on the background of the respondents, and it is included to characterize the demographics of the survey sample. The second set begins by asking respondents to think about the most complex system they have been involved in the last 10 years. That system will be used as a reference to answer the rest of the questions in the survey. The questions in the second set focus on identifying

the characteristics of the selected system and the disciplines, approaches and terms used in the corresponding project. To profile the selected system in terms of the SoS characteristics discussed in Section 3.1, questions in this set used a slider widget to allow to allow respondents to select a continuous value between 0 and 5 for each characteristic. The third and final set of questions collect data about the architecture practices used and problems observed by the respondent while working on the selected system. The complete questionnaire is available in the study replication package[3]. This is the version of the questionnaire that was actually used for the survey, following a few iterations of pilot studies for its improvement, as discussed further in Section 3.3.

3.2.2 Obtaining data. The survey was developed and distributed through the *Qualtrics.XM* platform [37]. The target population of this study was experts with a background in Software Engineering, Systems Engineering or both, who have been involved in engineering projects for systems with SoS characteristics. This background and experience makes the target population rather specific; hence, it is not trivial to obtain a representative sample. Consequently, we followed a non-probabilistic sampling that combined convenience and snowballing sampling [27]. More specifically, an invitation to participate and to further disseminate the survey was distributed by email across the personal networks of the authors to contacts who worked in industries like the ones mentioned in the study goal. In addition, the survey was promoted through posts on relevant LinkedIn groups and other social media platforms, the mailing list of INCOSE's *Systems and Software-Interface working group* which had 54 members, and through publicity fliers at the INCOSE's International Workshop (held on January 25th/2020). Moreover, the first author manually searched for publicly available email addresses of practitioners who were likely to fit the inclusion criteria, based on LinkedIn profile and group membership information. As a result, 281 additional contacts were identified and then invited to participate in the survey, through a series of personalized emails which described the survey, the average completion time, and linked to the questionnaire itself. Two follow-up reminder messages were sent to the potential respondents between mid-January and mid-February, 2020. This resulted in a total of 76 responses.

3.2.3 Analyzing the data. The data obtained from the survey were analyzed with a combination of quantitative and qualitative methods. The responses to the open-ended questions were analyzed using *Qualitative Content Analysis* (QCA) [16] following an inductive approach, which involved open coding, creating categories, and abstraction. The responses to closed-ended questions, on the other hand, were analyzed following quantitative analysis methods, including data visualization and statistical analysis [43]. More specifically, descriptive statistics, including frequencies and percentages, were used to describe the characteristics of the sample, and the relationships between the variables. Since there were no hypotheses to test, no statistical testing was employed.

3.3 Threats to validity

In the following we discuss the perceived threats to the validity of this study and the steps taken to mitigate them. We use Runeson and Höst [38] as a guide for this purpose. We note that, as the nature of this study is exploratory and did not investigate causal relationships, it is not subject to internal validity threats.

Construct validity. Construct validity refers to the degree to which the operational measures, in this case the online survey, reflect what the researchers have in mind and are consistent with the research questions. To improve the validity of the study in this respect, we piloted two consecutive versions of the survey, each with a different set of respondents, for a total of 7 respondents. These respondents were selected according to the survey goal (see Section 3.1), and could be described as experienced practitioners and researchers with background in Systems Engineering (3), and both Systems Engineering and Software Engineering (4). The pilot survey respondents provided key feedback on the wording and the consistency of the questions, particularly the ones that involved Systems Engineering-related terms. This allowed us to minimize potential miscommunication issues. The authors have also iteratively refined the study design to ensure that all aspects of the study were clear prior to commencing the survey.

External validity. External validity is concerned with the degree to which the findings can be generalized from the sample to the population. The non-probabilistic sampling design used for data collection is a potential threat for the external validity of the study. For instance, there is a risk of a biased sample, which is not representative of the target population, or with a dominant participation of a certain sector. To mitigate this threat, the survey was distributed not only through the personal networks of the authors, but also through organizations and social media platforms that address systems and software engineers from different application domains. The demographic information of the participants reported in Section 4.1, including the application areas of the organization they belong to, attests to the representativeness of our sample. However, we also have to acknowledge that our findings cannot be generalized beyond the population our sample represents, e.g. in systems that do not exhibit SoS characteristics or in application domains outside those in Fig. 3.

Reliability. Reliability refers to the extent the data and the analysis are dependent on the specific researchers. The mature and generally-accepted guidelines defined by Kitchenham and Pfleeger [25] were followed for this purpose; the analysis of the respondents' responses can also be verified externally by means of the available replication package. Given the exploratory nature of this study, we did not use advanced statistics-based analysis approaches; instead, we used a combination of quantitative and qualitative methods. To mitigate researcher bias in this process, three of the four authors were involved in the quantitative data analysis for consensus-building purposes; these authors also came to an agreement about the interpretations drawn from the analysis. Finally, the same authors were involved in the qualitative data analysis: the first author performed it using the QCA methodology [16], while its outcome was validated for consistency by the other two.

[3] https://figshare.com/s/5719c26fc74842ddc7ba

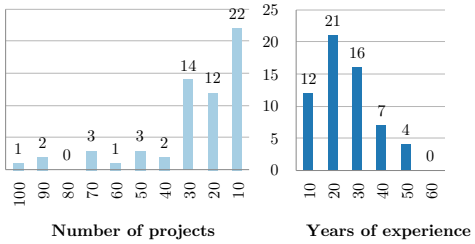

Figure 1: *Right:* years of experience of the respondents. *Left:* number of engineering projects respondents worked on

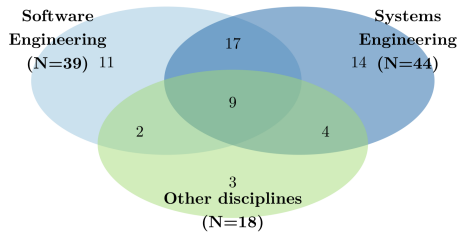

Figure 2: Distribution of the selected engineering practice areas across the survey respondents as an Euler diagram

4 RESULTS

4.1 Demographics

A total of 76 responses were collected between December 20, 2019 and February 25, 2020. From this initial data set, 16 responses were excluded before proceeding with the analysis: 15 responses were incomplete and 1 response was from a respondent that indicated zero years and projects as experience. Hence, the final data set contains 60 responses from practitioners with a mean of 20.8 years of experience who have participated on average in 24.4 engineering projects, shown in Fig. 1. Three outlier respondents claimed experience of more than 90 projects each, which makes the standard deviation of the number of projects nearly double that of the years of experience (22.8 vs 11.19).

Figure 2 shows the engineering practice areas of the respondents. The areas most frequently reported were *Systems Engineering* (73%) and *Software Engineering* (65%). Respondents were able to select more than one practice area. The Euler diagram shows that a total of 57 respondents (95%) identified themselves as *Systems Engineers* (30%), *Software Engineers* (22%) or both (43%). The remaining 3 respondents, added areas related to systems or software engineering, as entries in 'Other': *Computer systems designer*, *Software Systems Engineering* and *Software-reliant Systems Engineering*. All the selected responses came from practitioners that fit the target population described in the study goal.

The respondents' organizational affiliations are shown in the top part of Fig. 3. Multiple selections were allowed. Many respondents (78%) identified with companies. Fewer respondents (around 20%) also identified their organizations as research institutes, universities, or government agencies. The bottom part of Fig. 3, shows the application areas of the respondents' organizations. The results of this survey are representative of domains such as transportation

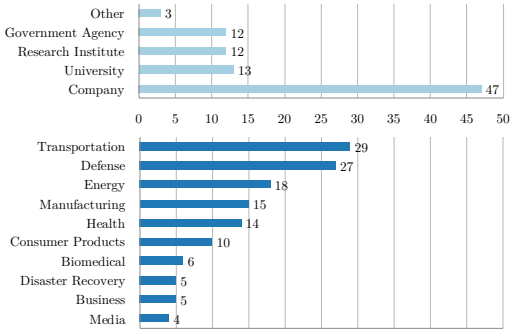

Figure 3: Organizations that respondents are affiliated with. *Top:* organization type; *Bottom:* application areas of the organizations

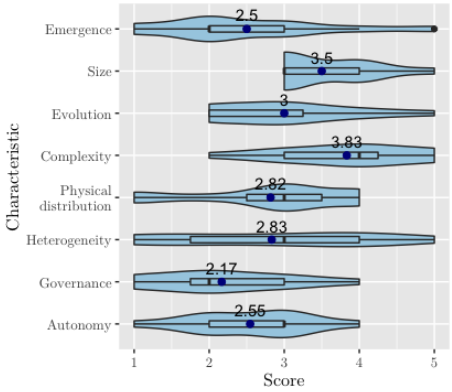

Figure 4: Distribution of project characteristics as violin plot

(48%), defense (45%), energy (30%) and manufacturing (25%). This aligns with the study's context: application domains where SoS architecting requires an interplay between the systems and software engineering disciplines.

4.2 Reference projects

As described in Section 3.2.1, respondents were asked to select a reference system to complete the survey and profile the characteristics of that system. Figure 4 shows the distribution of response values for each characteristic. Values for the Size, Complexity and Heterogeneity characteristics show that the sample of systems explored in this study represents large, highly complex systems, whose constituents are very heterogeneous. The response values for the Emergence, Governance and Autonomy characteristics are spread throughout the value range, indicating projects with different degrees of control on the constituent systems.

Respondents were also asked whether the selected project self-identified its system as a System of Systems. Of the 60 responses, 37 used the label 'SoS' to describe the system, whereas 23 used other terms such as *Complex System*, *Software Intensive System* and *Embedded System*. **However, by comparing the distribution of the characteristics of the systems that self-identify as SoS with the ones that do not, shown in Fig. 5, there are only minor**

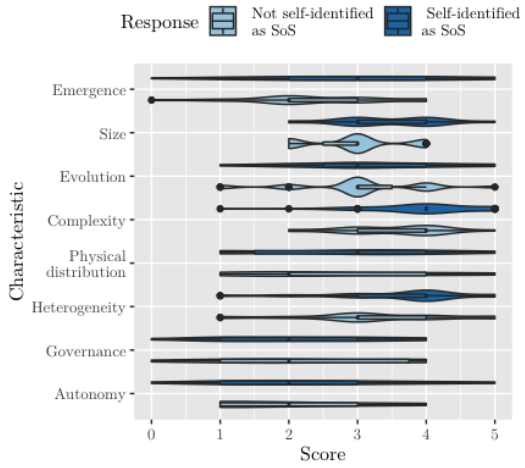

Figure 5: Distribution of project characteristics: systems self-identified as SoS vs. not self-identified ones

differences (Finding 1), and they appear mostly in the distributions of *Size*, *Evolution* and *Complexity* values. This appears to justify our decision to survey practitioners irrespective of whether they use the term 'SoS' for their systems, and instead focus on ensuring the presence of SoS characteristics.

Furthermore, it is worth noting that out of the 37 systems self-identified as SoS, only 10 applied well-known SoS-specific guidelines (e.g. *DoD guidelines for SoS* [34] or the *ISO/IEC/IEEE 21839–40* [24]), and none made reference to any of the guidelines or frameworks proposed, in recent years, by research in the field of SoS (e.g. [4, 31]). **This suggests a low adoption by practitioners of SoS-specific guidelines, and an even lower adoption of the research produced in the domain of SoS (Finding 2).**

4.3 Questions related to RQ1

4.3.1 Requirements at system and software level. Q 3.1 of the survey, which focused on the architecting phase of *Analysis*, asked respondents *how system-level requirements and software-level requirements were related to each other* in the system selected by them during the second set of questions (see Section 3.2.1). The responses are summarized in Table 1. In most cases, software-level requirements are linked to system requirements (71%), either by being explicitly derived from them (39%) or based partially on them (32%). Figure 6, however, shows that there are no significant differences in the characteristics between the systems developed following each approach. It appears that the choice of approach is not related to the characteristics of the system; further investigation is necessary.

In a follow-up question, respondents were asked an open-ended question about the *problems posed by the way system-level and software-level requirements were related to each other*. Six codes emerged as the most prominent ones after applying QCA to the responses. Two of the most frequent codes, Teams Coordination Challenges and Assumptions About Other Disciplines, seem to be related, in the sense that the former could be reinforcing the latter. That is to say, due to the challenges of coordination and communication between interdisciplinary teams, there could be assumptions

Table 1: Frequency of the responses given to Q 3.1: relation of system– and software–level requirements

Response	Frequency
Software-level requirements were derived from system-level requirements	23
Software-level requirements were elicited by taking into consideration, among others, system-level requirements	19
Software-level and system-level requirements were elicited independently by separate teams	7
Software-level and system-level requirements were elicited simultaneously by the same team	5
Other	6

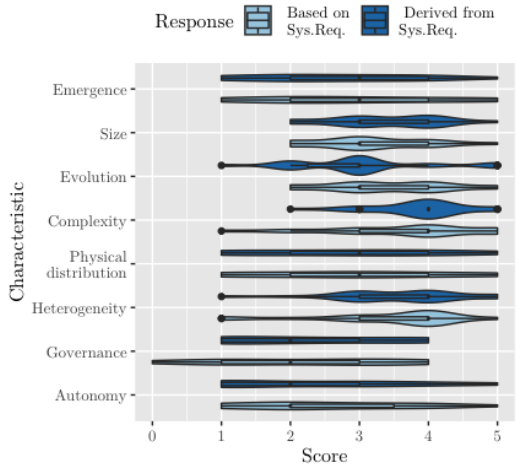

Figure 6: System characteristics of the two most common responses to Q 3.1

about how the other party would address the requirements, instead of agreements. Incomplete System Requirements (which includes vague or poorly documented requirements) and Lack of Domain Knowledge from the Software Engineers correspond to the problems that lead to misinterpretation of system-level requirements at the software architecture level. **These two problems (incomplete systems requirements and lack of domain knowledge) are particularly important: for most of the systems considered in this survey, software-level requirements are derived from or based on system-level requirements (Finding 3)**, as discussed above. The remaining two codes are Lack of System-level Perspective by the teams and organizations involved, and perhaps even more interestingly Interdisciplinary Differences resulting in conflicting or over-restrictive requirements coming from the system level.

4.3.2 Architectural decisions at system and software level. Q 3.2, related to the architecting activity of *Synthesis*, asked respondents *how the architectural decisions were taken for the selected project, at system and software level*. Table 2 shows that most of the responses (77%) are nearly evenly distributed between two opposite approaches: Decisions are taken separately by independent

Table 2: Frequency of the responses given to Q 3.2: how were architectural decisions at different levels taken

Response	Frequency
System-level and software-level architectural decisions were taken independently, by separate teams	24
System-level and software-level architectural decisions were taken, together, by the same team	21
Architectural decisions were taken at system level only (there was no intentionally designed software architecture)	4
Other	10

systems-engineering and software engineering teams (41%), or taken together by an interdisciplinary team (36%). **The high ratio of systems where architectural decisions were taken independently by teams from the two disciplines is noteworthy, because intuitively, decisions taken independently could be problematic in many cases. (Finding 4)**

Q 3.3 asked about *the problems posed by the way architectural decisions were taken at both system and software level.* Applying QCA to the responses shows that the most common problems can be classified as related to: *integration* (70% of responses in total), *documentation* (53%) and *operation* (43%). The problems reported by the respondents as 'Other' and coded as Unsatisfied requirements and Bugs/issues caused by under-scrutinized complex interactions were counted as operation-related problems, and included in the above percentage. Likewise, Third-party components inconsistencies and Poor documentation, also reported by the respondents as other problems, were counted as integration and documentation-related problems, respectively. An observation that requires further investigation is that respondents that selected both the SE and SWE practice areas report more problems with the architectural decisions (49% of all cases reported) compared to respondents with other backgrounds. It may be that their understanding and experience in both practice areas allows them to diagnose issues that otherwise would go unnoticed until much later; this is the subject for a separate, future study.

Another observation is that **both of the most frequently reported approaches for making architectural decisions, i.e. in separate or interdisciplinary mixed teams, report a very high rate of integration problems (Finding 5)**. As shown in Fig. 7, both approaches report similar frequency of documentation problems, but surprisingly operational problems were reported for only 33% of systems where teams from different disciplines worked separately, compared to 48% of systems using interdisciplinary teams. This may be simply due to reduced visibility of the overall operational situation in compartmentalized teams.

4.3.3 Architecture evaluation at system and software level. The questions related to the architecting activity of *Evaluation* asked *how system-level (Q 3.4) and software-level (Q 3.5) architectural decisions were evaluated with respect to functional and non-functional requirements*, and *what problems were posed by combining these two evaluation approaches* (Q 3.6). The responses to Q 3.4, summarized in Table 3, show that simulations and prototyping were the most

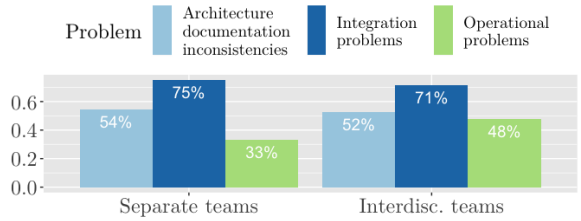

Figure 7: Problems reported given the approach followed for architectural decision making, normalized (Q 3.3)

Table 3: Frequency of the responses given to Q 3.4: evaluation of architectural decisions at system level

Response	Frequency
Simulation-based evaluation	21
Prototype-based evaluation	17
Evaluation in production	16
Formal/mathematical evaluation	11
The architectural decisions were not evaluated	4
Do not know	7
Other	14

Table 4: Frequency of the responses given to Q 3.5: evaluation of architectural decisions at software level

Response	Frequency
Software architecture was evaluated against the system requirements, after the system requirements and architecture were established	20
Software architecture was considered as implicitly evaluated by the the system architecture validation	12
Software architecture was evaluated evaluated against a set of requirements that are not related to the system requirements	7
There was no architecture evaluation in the project	4
Do not know	9
Other	8

frequently reported approaches for system-level architectural decision evaluation. The responses to Q 3.5, summarized in Table 4, on the other hand, show that in most cases (53%), the evaluation of software-related architectural decisions were subordinated to the system-level architecture: The evaluation was either based on system-level requirements (33%), or considered as implicitly evaluated by the system-level architecture validation (20%). **Software-related architectural decisions, from the evaluation standpoint, appear to have lower priority compared to the system-related decisions. (Finding 6)** From these results, it is also worth noting that approaches that would be expected to be more frequent, such as the *scenarios-based* [11] ones, were mentioned as *Other* approaches by a limited amount of respondents.

In the Euler diagram of Fig. 8, there appear two phenomena on how the different system-level evaluation approaches were combined. First, for 17% of the systems, the system-level architectural

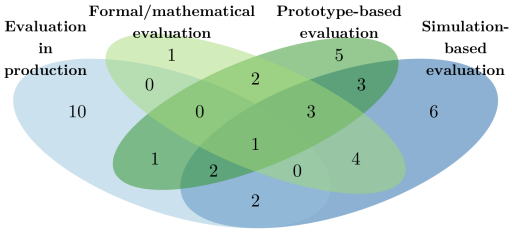

Figure 8: Distribution of the selected approaches for System-level architecture evaluation

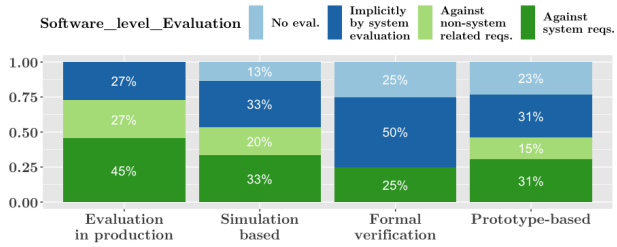

Figure 9: Distribution of Software–level architecture evaluation approaches given the followed System-level architecture evaluation approaches

decisions were evaluated *only when the system became operational* (i.e. the entire SoS running in a production environment). Second, in most cases, *a combination of two or more system-level evaluation approaches took place*. Further related phenomena can be identified in Fig. 9, which shows the distribution of the approaches followed for the evaluation of software-level architectural decisions, given the approaches followed for the system-level ones. While Evaluation in Production must include some evaluation of the software, for each of the other three evaluation approaches there were a portion of projects that reported doing no evaluation of software architecture decisions. Furthermore, **in most of the systems that used a formal approach for the validation of their architecture, there was no separate evaluation at software-architecture level, or it was considered as implicitly evaluated. (Finding 7)**

Regarding the problems arising by combining system-level and software-level evaluation approaches (Q 3.6), the QCA analysis categorized the 22 free-text responses in four codes: Architecture Patching, Insufficient Evaluation, Evaluation Criteria and Evaluation Complexity. It is interesting that the responses linked to the most frequent problem category, Architecture Patching, were not related to the evaluation process per se, but to the effort required to address the issues found by the evaluation. Some responses suggest that a major part of the cost of these architecture patches is due to the lack of clarity about which level (system or software) to make the changes. Moreover, it is noteworthy that the architectural evaluation of the systems linked to these responses was either performed late in the process (in production) or through throw-away prototypes.

A number of respondents also report Insufficient Evaluation due to informal evaluation procedures, insufficient attention, or lack of learning from the evaluation. Finally, the categories Evaluation Complexity and Evaluation Criteria both identify challenges in performing an evaluation. The first includes responses that describe how the complexity of the architectural descriptions and the processes involved in the system operation makes the evaluation process equally complex. Evaluation Criteria includes responses that describe the challenges on their definition given the need for addressing both system- and software-level elements.

4.4 Questions related to RQ2

The results presented in this section concern the problems posed by the interplay between SWE and the SE disciplines while addressing two of the most distinctive characteristics of an SoS [28]: *Emergent Behavior* of an SoS and the *Independence* of its constituents. These questions were presented only to the respondents who scored these characteristics higher than 0 for their reference system.

4.4.1 Emergent behavior. Q 4.1 asked respondents whose system presented some degree of emergence (54 out of 60), *whether this characteristic was an architectural concern, and if so, at which levels (system or software level, or both) it was addressed.* As seen on Table 5, only 13% of this subset reported that emergence was not an architecture concern, whereas in most of the cases (43%) emergence was addressed as an architectural concern at both system and software level architectures. Only in a small fraction of the cases emergence-related architectural concerns were addressed exclusively at either the system or software level (7% and 4% respectively).

In the follow-up question (Q 4.2), respondents were asked an open-ended question about *the problems posed by the way that emergence was addressed.* QCA was performed on the responses and 3 codes linked to requirements-related problems emerged as the most prominent ones. The first, Modeling and Simulation Limitations, includes quotes on how the limitations of the modeling approaches for highly complex systems and the imperfectness of the simulation approaches make it difficult to anticipate and address undesired emergent behaviors. The second code, Subsystems-related Emergence Causes includes quotes that describe problems at the Constituent System level related to undesired emergent behaviors, particularly failures and timing problems with low level hardware components. Together, the quotes linked to these two codes suggest that **low level hardware components, despite being one of the root causes of undesired emergent behavior, are sometimes not considered in the architecting process of SoS. Furthermore, this could be due to limitations of the modeling and simulation approaches, where such low-level components are not considered due to the granularity level of such approaches. This is particularly important since simulation is the most common approach to evaluate architectural decisions at system level (Finding 8)**, as discussed in Section 4.3.3.

Finally, quotes linked to the third code, Time and Resources Problems, describe both resource-related causes (insufficiently allocated time and resources) and consequences of undesired emergent behavior (added costs, rework effort).

4.4.2 Autonomy/Independence. The interdisciplinary problems between SE and SWE related to the characteristic of independence of SoS constituents are explored through questions Q 4.3, Q 4.4 and

Table 5: Frequency of the responses given to Q 4.1: desired or undesired emergent behavior as an architectural concern, and means to address it

Response	Frequency
Emergent behaviour was addressed by both system-level and software-level architecture	23
Emergent behaviour was a concern, but it was not addressed by the architecture	10
Emergent behaviour was not a concern	7
Emergent behaviour was addressed only by system-level architecture	4
Emergent behaviour was addressed only by software-level architecture	2
Other	8

Q 4.5. These problems are explored from the perspective of the interfaces and interface specifications of the constituent systems. More specifically, Q 4.3 asked the respondents about *the existence of interface specifications of the system's constituents and the details provided in the specifications.* All 60 respondents scored the characteristics of Autonomy or Governance (corresponding to Operational and Managerial independence, respectively) above zero for their reference systems, and so these questions were presented to all respondents. However, only 28 respondents provided information about the details provided in the interface specifications and only 3 respondents reported cases where all specification elements (syntactic, behavioral and QoS) were present.

Q 4.4 asked respondents about *the kind of problems posed (if any) at system and software architecture level by the integration of independent constituents.* Out of the 60 respondents, 13 identified integration problems at software level due to loose or inconsistent specifications. 17 identified problems at system level due to incomplete specifications; 7 of those responses did not offer any further clarification, while the remaining 10 responses reported incomplete specifications, irrespective of what and how many interface specification types were developed.

In the follow-up question (Q 4.5), respondents were asked an open-ended question about *the problems posed by the changes or evolution of the independent constituents, given the way their interfaces were provided.* QCA was performed on the 24 responses and one of the most common codes was Incomplete Interface Specifications. Despite its prominence, Incomplete Interface Specifications can be seen as a cause of problems as the constituent systems change or evolve, rather than a problem per se. However, since 50% of the respondents reported some kind of problems due to incomplete or inconsistent interfaces (Q 4.4), the results of the qualitative analysis on this question suggest that **the unclear/incomplete interface specification problems faced during the system integration remain an issue as the independent constituents evolve. This suggests that unclear/incomplete interface specification problems could be a cause of technical debt in SoS. (Finding 9)** Another common code, *Configuration Management*, provides further insight into problems that arise as the constituent systems evolve. These quotes describe how **inadequate configuration management of interface specifications and the lack of policies for their life-cycle lead to specifications that were out-of-sync with reality. Moreover, communication of the specification changes to the interested parties might not be considered as part of the configuration management process. (Finding 10)** The remaining two codes reflect issues related to the management of change: how resources are allocated (Change Costs) or the way the changes are approached (Hardware–Software Changes Balance).

5 DISCUSSION

This survey collected data about the problems posed by the interplay between the SE and SWE disciplines during SoS architecting in practice. Each respondent was asked to characterize their reference system along 8 dimensions. We discovered that there were only minor differences in these characterizations between respondents that labeled their system as SoS and those that did not. (**Finding 1**). This, together with the strikingly low adoption of SoS-specific guidelines, frameworks and tools for system development even in systems explicitly identifying as SoS (**Finding 2**), points to *a state of practice that seemingly fails to take advantage of the existing body of knowledge about SoS and the accruing benefits from it.* This gap between research efforts and industrial needs is something that this and other empirical studies of SoS and SE/SWE interplay, which are scarce at the moment could and should contribute to address.

Looking at the research questions, RQ1 focused on SoS architecture analysis, synthesis and evaluation. The results show that software requirements are in most cases subordinate to system-level requirements. Although this is not surprising, our study highlighted major issues in the software part of the system when its requirements are derived or based on higher-level system requirements (**Finding 3**); consider for example, the cycle of interdisciplinary coordination problems (also reported in other studies e.g., in [39]) and assumptions made about each other's discipline leading to missed requirements. In other cases, misinterpretations of the system-level requirements are reported, either due to their incompleteness or a lack of domain knowledge of the software engineers. Problems like the lack of domain knowledge, or the lack of 'system-perspective' have been recognized since the earliest related literature [18, 42]. However, the fact that software engineers are perceiving the incompleteness of system-level requirements as a cause of misinterpretation urges *further research towards requirement specification approaches that are more balanced for both disciplines.*

Considering how architectural design decisions at the system and software level are taken, we found that in many systems (41%) such decisions were taken by separate teams working independently at each level. Intuitively, this approach is more likely to be linked to problems in a project, however it turns out that the distribution of integration and documentation issues—the most frequent ones overall—are almost identical when compared to the systems whose design decisions were taken by interdisciplinary teams working together (**Finding 4**). *This suggests that there may be forces common to both approaches that cause these problems, with further study warranted to identify the root causes of the problems.* The high percentage of integration problems (above 70%) reported (**Finding 5**) *further reinforces our previous point about the importance of bridging the gap between research and industry.* For instance,

a significant number of studies on SoS interoperability have already been reported [10] but barely ever used. Furthermore, more than 50% of systems using either architecture decision approach (interdisciplinary vs. separate), report documentation-related problems. This suggests that even with a closer cooperation between the two disciplines, the consistency of the architecture documentation is still problematic. It might therefore be worth *developing an architecture description language or framework for better capturing details at both levels, while reconciling the terminology differences.*

Turning to the evaluation phase of the architecting process, we found that system-level architectural decisions are evaluated, in most cases, through a combination of prototyping and simulation approaches. However, the architectural decisions at software level, despite being taken independently in many cases, are almost exclusively validated against system-level requirements (**Finding 6**), or even considered as implicitly evaluated, particularly in systems evaluated using a formal approach (**Finding 7**). This suggests that in most SoS governed by a systems engineering process, the evaluation process might not be considering the software requirements (esp. non-functional requirements) when they are not explicitly linked to a higher-level system requirement. This is worth exploring further through studying *the implications of not integrating software-specific architecture evaluation methods in higher-level system evaluation processes, as well as ways to integrate such evaluation approaches.*

We also explored the interdisciplinary problems linked to the independence of SoS constituents and the behaviors that emerge from the constituents' cooperation [28]. The results of the survey show very few problems explicitly linked to the interplay of SE and SWE disciplines when dealing with undesired emergent behavior. However, the results reveal another issue to investigate further: *failures of low-level hardware subsystems, which are not considered in the modeling or simulation environments due to their granularity level, are a common cause of undesired emergent behavior* (**Finding 8**). Furthermore, it is worth exploring *to what extent software subsystems are causing emergent behavior, because (like low-level hardware subsystems) they might not be included in higher-level system models and simulations.*

Finally, the characteristic of constituent system independence was explored from the perspective of the interfaces provided for the integration of the independent constituents. We found that the problem of incomplete specifications is reported both early in the integration process and during system evolution. This seems to be a common cause of technical debt in SoS projects (**Finding 9**). Interestingly, another problem identified in this study could be the key to address this kind of technical debt: insufficient interface specifications management (**Finding 10**). As suggested by some of the respondents, interface management in SoS should not only become mandatory, but it needs to go beyond tracking and controlling documentation changes to consider the full life cycle of interfaces and the distribution/communication of changes to the involved parties. *An extension to the interface management processes defined in well-known engineering bodies of knowledge [13, 36], considering the aforementioned life cycle and distribution elements, and the hardware-software interfacing problems discussed in previous works (e.g. [17]), is worth exploring further.*

Overall, we feel that *software engineering as a discipline*, when applied in the context of larger systems and particularly SoS, *must address the interplay with other disciplines and move towards using standardized practices.* We would like to note in particular that most of the works that explore the interdisciplinary problems between SE and SWE, like the ones discussed in Section 2, actually come from the SE community. For instance, although the *Systems Engineering Body of Knowledge* (SEBOK) [36] acknowledges the interdisciplinary challenges with the software engineering discipline, the *Software Engineering Body of Knowledge* (SWEBOK) [9] makes very few references to other engineering disciplines. The research topics derived from this study and other similar empirical studies in the field could contribute significantly to this underrepresented aspect of the software engineering discipline.

6 CONCLUSIONS

The Systems Engineering (SE) and Software Engineering (SWE) disciplines are highly interdependent in the development of modern SoS in industries like defense, automotive, energy and health care. However, the literature suggests that the gaps or inconsistencies between the architecting approaches followed by each discipline often cause system-wide problems in the resulting systems. This is, to the best of our knowledge, the first empirical study that explores the problems experienced by practitioners in the architecting process of SoS when both disciplines are involved. For this purpose we designed, piloted and carried out an online questionnaire-based practitioner survey. We obtained a total of 60 pertinent responses.

The results of this study revealed predominant problems related to three architecting artifacts that could benefit from further interdisciplinary research, namely *requirements specifications* across system and software level, *architectural description languages* for the documentation of the system, and *interface specifications* to enable the independent evolution of the constituent systems. In addition, the findings of this study also show persistent issues with SoS architecting resulting in integration difficulties and an apparent low priority given to the evaluation of software-specific architectural decisions. In combination with the reported very low adoption of the existing body of knowledge, such issues are a cause of concern and a call for further interdisciplinary research between SE and SWE. Last but not least, this study also uncovered a phenomenon linked to undesired emergent behavior worthy of further exploration: limitations of current system-level modeling and simulation approaches due to their exclusion of low-level components.

In addition to working on the open research issues identified in Section 5, in the future we plan to conduct further empirical studies oriented towards the identification of SE/SWE architecting harmonization practices in SoS and their relation to the problems identified in this study.

ACKNOWLEDGEMENTS

This work was supported by ITEA3 and RVO under grant agreement No. 17038 VISDOM (https://visdom-project.github.io/website/).

The authors would also like to thank the participants to the pilot studies of this work for their invaluable help: Rochus Keller, Osasai Macauly, Sally C. Muscarella, Philippe Krutchen, Lars de Groot, Remco Poelarends and Sarah Sheard, and the anonymous reviewers for their comments.

REFERENCES

[1] 2011. *COMPASS: Comprehensive Modelling for Advanced Systems of Systems.* http://www.compass-research.eu/

[2] 2011. *DANSE: Designing for Adaptability and Evolution in System-of-Systems Engineering.* http://www.danse-ip.eu/

[3] 2013. *AMADEOS: Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems.* http://amadeos-project.eu/

[4] Arun Babu, Sorin Iacob, Paolo Lollini, and Marco Mori. 2016. AMADEOS Framework and Supporting Tools. In *Cyber-Physical Systems of Systems*. Springer, 128–164.

[5] W Clifton Baldwin and Brian Sauser. 2009. Modeling the characteristics of system of systems. In *2009 IEEE International Conference on System of Systems Engineering (SoSE)*. IEEE, 1–6.

[6] Victor R Basili. 1992. *Software modeling and measurement: the Goal/Question/Metric paradigm.* Technical Report.

[7] J Boardman, S Pallas, BJ Sauser, et al. 2006. *Report on system of systems engineering, final report for the office of secretary of defense.* Technical Report. Hoboken, NJ: Stevens Institute of Technology.

[8] Barry Boehm and Jo Ann Lane. 2007. Using the incremental commitment model to integrate system acquisition, systems engineering, and software engineering. *CrossTalk* 19, 10 (2007), 4–9.

[9] Pierre Bourque, Richard E Fairley, et al. 2014. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press.

[10] Héctor Cadavid, Vasilios Andrikopoulos, and Paris Avgeriou. 2019. Architecting Systems of Systems: A Tertiary Study. *Information and Software Technology* (2019).

[11] Paul Clements, R Kazman, and M Klein. 2002. *Evaluating software architectures: methods and case studies.* Addison-Wesley.

[12] James A Crowder, John N Carbone, and Russell Demijohn. 2015. *Multidisciplinary systems engineering: Architecting the design process.* Springer.

[13] DAU. 2004. *Defense acquisition guidebook.* Defense Acquisition University.

[14] Michael J de C Henshaw. 2016. SYSTEMS OF SYSTEMS, CYBER-PHYSICAL SYSTEMS, THE INTERNET-OF-THINGS… WHATEVER NEXT? *Insight* 19, 3 (2016), 51–54.

[15] Richard De Neufville et al. 1994. The baggage system at Denver: prospects and lessons. *Journal of Air Transport Management* 1, 4 (1994), 229–236.

[16] Satu Elo and Helvi Kyngäs. 2008. The qualitative content analysis process. *Journal of advanced nursing* 62, 1 (2008), 107–115.

[17] Richard E Fairley. 2019. *Systems Engineering of Software-enabled Systems.* John Wiley & Sons.

[18] Richard E Fairley and Mary Jane Willshire. 2011. Teaching systems engineering to software engineering students. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 219–226.

[19] Donald Firesmith. 2010. *Profiling systems using the defining characteristics of systems of systems (SoS).* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.

[20] Michael Gagliardi, WG Wood, J Klein, and J Morley. 2009. A uniform approach for system of systems architecture evaluation. *CrossTalk* 22, 3-4 (2009), 12–15.

[21] Alex Gorod, Brian Sauser, and John Boardman. 2008. System-of-systems engineering management: A review of modern history and a path forward. *IEEE Systems Journal* 2, 4 (2008), 484–499.

[22] Christine Hofmeister, Philippe Kruchten, Robert L Nord, Henk Obbink, Alexander Ran, and Pierre America. 2005. Generalizing a model of software architecture design from five industrial approaches. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*. IEEE, 77–88.

[23] ISO/IEC/IEEE 15288:2015 2015. *ISO/IEC 15288 Systems and software engineering - System life cycle processes (also: IEEE Std 15288-2008).* Standard. International Organization for Standardization, Geneva, CH.

[24] ISO/IEC/IEEE 21839:2018(E) 2018. *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.

[25] Barbara A Kitchenham and Shari L Pfleeger. 2008. Personal opinion surveys. In *Guide to advanced empirical software engineering*. Springer, 63–92.

[26] John Klein and Hans van Vliet. 2013. A Systematic Review of System-of-Systems Architecture Research. In *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA '13)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/2465478.2465490

[27] Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. 2015. Guidelines for conducting surveys in software engineering. (2015).

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

[29] Mark W Maier. 2006. System and software architecture reconciliation. *Systems Engineering* 9, 2 (2006), 146–159.

[30] Mark W Maier. 2009. *The art of systems architecting.* CRC press.

[31] L Mangeruca, R Passerone, C Etzien, T Gezgin, T Peikenkamp, M Jung, A Alexandre, R Bullinga, S Imad, E Honour, et al. 2013. Designing for adaptability and evolution in system of systems engineering-DANSE Methodology V2. *The Seventh Framework Programme* (2013).

[32] Brian Mekdeci, Nirav Shah, Adam Michael Ross, Donna H Rhodes, and Daniel E Hastings. 2014. Revisiting the Question: Are Systems of Systems just (traditional) Systems or are they a new class of Systems? (2014).

[33] Gerrit Muller. 2012. Validation of systems engineering methods and techniques in industry. *Procedia Computer Science* 8 (2012), 321–326.

[34] Office of the Under Secretary of Defense - AT&L 2008. *Systems Engineering Guide for Systems of Systems.* Office of the Under Secretary of Defense - AT&L.

[35] Art Pyster, Rick Adcock, Mark Ardis, Rob Cloutier, Devanandham Henry, Linda Laird, Michael Pennotti, Kevin Sullivan, Jon Wade, et al. 2015. Exploring the relationship between systems engineering and software engineering. *Procedia Computer Science* 44 (2015), 708–717.

[36] Art Pyster, David H Olwell, Nicole Hutchison, Stephanie Enck, James F Anthony Jr, Devanandham Henry, et al. 2012. Guide to the systems engineering body of knowledge (SEBoK) v. 1.0. 1. *Guide to the Systems Engineering Body of Knowledge (SEBoK)* (2012).

[37] UARK Qualtrics. 2019. Qualtrics research suite. *Copyright© 2019* (2019).

[38] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131.

[39] Sarah Sheard, Rita Creel, John Cadigan, Joseph Marvin, Leung Chim, and Michael E Pafford. 2018. INCOSE Working Group Addresses System and Software Interfaces. *INSIGHT* 21, 3 (2018), 62–71.

[40] Sarah Sheard, Michael E Pafford, and Mike Phillips. 2019. Systems Engineering–Software Engineering Interface for Cyber-Physical Systems. In *INCOSE International Symposium*, Vol. 29. Wiley Online Library, 249–268.

[41] Raghu Singh. 1996. International Standard ISO/IEC 12207 software life cycle processes. *Software Process: Improvement and Practice* 2, 1 (1996), 35–50.

[42] Ian Sommerville. 1998. Systems engineering for software engineers. *Annals of Software Engineering* 6, 1-4 (1998), 111–129.

[43] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering.* Springer Science & Business Media.