What We Know About Software Architecture Styles in Continuous Delivery and DevOps?

Maya Daneva(✉) and Robin Bolscher

University of Twente, 7522NH Enschede, The Netherlands
m.daneva@utwente.nl, r.bolscher@student.utwente.nl

Abstract. This paper takes a software architect's perspective to DevOps/CD and attempts to provide a consolidated view on the architecture styles for which empirical publications indicate to be suitable in the context of DevOps and CD. Following techniques from the evidence-based software engineering paradigm, we set out to answer a number of research questions pertaining to (1) the architecture characteristics important in DevOps/CD projects according to published literature, (2) the architectural styles found to work well in this context, (3) the application domains in which architecture characteristics and styles were evaluated, and (4) the empirical method being used by researchers on this topic. We applied a research protocol grounded on well-established systematic literature review guidelines, and evaluated sources published between 2009 and 2019. Our results indicate that (a) 17 software architecture characteristics are beneficial for CD and DevOps adoption, (b) micro-services are a dominant architectural style in this context, and (c) large-scale organizational contexts are the most studied, and (d) qualitative approaches (case study based) are the most applied research method.

Keywords: Software architecture · Continuous delivery · Continuous integration · DevOps · Deployability · Micro-services · Systematic literature review

1 Introduction

Today, many businesses in the IT industry are embarking on DevOps and Continuous Delivery (CD). This interest in DevOps/CD is traceable to organizations' motivation to increase their abilities to deliver software fast and predictably well. The growing adoption of the DevOps and CD concepts is however not free of problems. For example, a 2017 systematic mapping study of literature [14] on CD challenges reports 40 problems discussed in scientific publications. The present article follows up on one of these problems, namely the use of unsuitable architecture in CD (and in DevOps) contexts. We felt intrigued to know what so far has been published on the qualities of suitable architectures for DevOps/CD. Our motivation to consolidate the published knowledge on this topic is based on the observation that although DevOps and CD have been employed massively for more than 5 years and much guidance has been published on how to implement these concepts well, little has been done so far to elaborate on the linkage between DevOps/CD and architecture. Yet, as Bass states [15],

M. van Sinderen and L. A. Maciaszek (Eds.): ICSOFT 2019, CCIS 1250, pp. 26–39, 2020.
https://doi.org/10.1007/978-3-030-52991-8_2

the DevOps practices have implications for the software architects in the DevOps-adopting organizations.

This research aims at consolidating the published experiences regarding the architecture styles' fit and misfit to DevOps and CD context. If such a consolidated view of the published knowledge exists, researchers would know those areas that have enjoyed much research efforts and those that are under-researched. Moreover, if a map of the published empirical evidence is provided to software architecture practitioners, they could possibly be able to consider it when making architecture design decisions in DevOps and CD contexts.

Using the techniques of the evidence-based software engineering paradigm, we designed a systematic review protocol in order to identify and evaluate the empirical evidence published on this topic. Our research took two stages: in *stage 1*, we investigated the software architecture challenges experienced in DevOps/CD-adopting organizations. In *stage 2*, we focus on the architecture styles that support the DevOps/CD implementation. The results of *stage 1* have been reported at the ICSOF 2019 conference [16]. The results of *stage 2* are now reported in the present paper. Although the two research stages are complementary and grounded on the same review protocol [16] and, in turn, analyze the same pool of selected literature sources, in contrast to the ICSOFT 2019 conference paper [16], this paper treats different research questions and therefore reports new findings.

The paper is structured as follows. Section 2 provides definitions of the terms used in our research. Section 3 presents the purpose of this work. Section 4 presents our research questions and the research method used. Section 5 presents the results of our SLR. Section 6 discusses the results. Section 7 is on the possible risks of passing bias into our study. Section 8 is on related work. Section 9 summarizes our findings and discusses some implications for research and practice.

2 Definitions of Terms

For clarity, before elaborating on the scope and the research questions of this SLR, we present the definitions of the concepts that we use [16]. Software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both [17]. Continuous Delivery (CD) is a software engineering discipline in which the software is kept in such a state that in principle, it could be released to its users at any time [17]. The discipline is achieved through optimization, automatization and utilization of the build, deploy, test and release process. Furthermore, DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into 'normal' production, while ensuring high quality [15]. For the purpose of this work, we borrow Wood's definition of 'production' [4]: this is "any environment which is being used to perform valuable work" in an organization. As one could see from the definitions, CD and DevOps have overlapping goals. Both concepts serve companies to take full (end-to-end) advantage of Agile and Lean [14]. Since the two concepts are so similar, the effect they have on software architecture is expected to be very similar as well. This is why these two concepts are both included in our SLR.

3 Purpose

The purpose of this SLR is to identify and analyze the relationship between DevOps/CD and software architecture by using published empirical evidence regarding the architecture styles that support the implementation of CD and DevOps. For this purpose, we followed three areas of interest:

(1) characteristics of architecture that are important in DevOps/CD context,
(2) application areas in which these characteristics were identified or evaluated,
(3) empirical research method that was used by the authors of the published studies on this topic.

The first area concerns the non-functional requirements that, if met, render a software architecture beneficial for systems implemented by using DevOps/CD practices. The second area of interest concerns the contexts in which these non-functional requirements are deemed important according to published literature. We assume that not every application domain has been subjected to active research and, in turn, our knowledge of the non-functional requirements that a software architecture meets to support the implementation of DevOps/CD, may be fragmented or skewed. This assumption is justified by the observation that in many empirical studies on other software engineering phenomena, some application domains are more researched than others. Finally, the third area of interest concerns the research-methodological foundation of the published research studies, which would allow us to evaluate the realism of the published findings and their generalizability [18]. The results in each of these three areas are analyzed, focusing on how frequently our findings appeared in the selected set of papers and how they are framed in each paper. We tried to identify any inconsistencies in the results so that we can provide further knowledge gaps and lines for future research.

4 Research Questions and Method

Based on the purpose of our literature study, we set out to answer three Research Questions (RQs):

RQ1: What software architecture characteristics have been deemed important for enabling DevOps and CD, according to published literature?
RQ2: What applications areas have been reported in published literature concerning the important architectural characteristics found in the answer to RQ1?
RQ3: What research methods have been used in the published empirical papers used to answer RQ1 and RQ2?

To answer these RQs, we planned and executed a SLR, adopting the guidelines of Kitchenham et al. [19]. These were complemented with the guideline of Kuhrmann et al. [20]. We adapted these guidelines to this specific research as elaborated in our review protocol (Fig. 1).

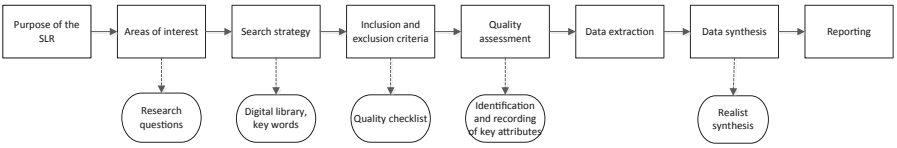

Fig. 1. Our research protocol.

For the purpose of our collection of possibly relevant articles, we explored the Scopus digital library by focusing on following string:

("software architecture" AND ("continuous delivery" OR "continuous deployment" OR "devops" OR "dev-ops" OR "dev ops")).

The search was carried out on May 14, 2019. It was applied to the Title, Abstract and Keyword sections of the Scopus digital library. Performing the search resulted in 39 papers from Scopus over a time span of 9 years (2009–2019). Our study selection process followed the inclusion and exclusion criteria listed below.

Inclusion Criteria

1. The paper treats DevOps/CD aspects as its core topic and discusses software architecture in this context;
2. The takes a practical point of view on the problems and/or solutions discussed (e.g. it is a case study or expert/practitioner experiences and opinions).

Exclusion Criteria

1. The paper presents no link to DevOps, CD or similar practices;
2. The paper is published before Jan 1, 2015;
3. The paper is purely theoretical;
4. The paper is a duplicate of a paper that was already found in Scopus;
5. The paper is not written in English.
6. The paper summarizes a workshop, a conference or another scientific event.

We would like to note that our process of the articles' selection was iterative and happened in multiple phases. The first application of the above list of criteria to titles and abstracts of the 39 papers resulted in a set of 23 papers that we deemed to fall in scope of this SLR. This reduction (from 39 to 23) was due to many duplicates. In the second iteration, we have read the 23 papers in detail and re-applied the inclusion/exclusion criteria. This ended up with 13 papers which we used in the data extraction and data synthesis stages of this SLR. The papers formed the following list of references: [1–13].

Once the paper selection was over, we focused on data extraction. This included carefully reading the whole text of each paper and keeping notes on the following pieces of information: countries of the affiliations of the authors, type of affiliation (industry or academic institution), explicit mentioning of software architecture characteristics, contextual settings in which the DevOps/CD concepts were implemented, application domain, explicit mention of research methodological sources and research

method used, treatment of validity questions while using a research method. The precise data have been coded, analyzed and compared by two authors. For the data synthesis, we followed the Pawson's realistic synthesis practices [21]. The authors worked independently from each other so that they the senior researcher does not expose the junior researcher involved in this study to possible bias due to the fact that the senior researcher knew some of the authors of the selected papers. The two authors consolidated their results and found no discrepancies in their analysis of the papers.

5 Demographics, Themes and Trends

This section reports our findings. Before presenting our answers to our three RQs, we first report some demographic information on the papers in our analyzed set.

First, in our set of 13, we have six papers authored by individuals working in companies [1–4, 12, 13], three papers authored by collaborators from companies and universities [6, 7, 9] working in industry-university projects, and four papers authored by academic researchers [5, 8, 10, 11]. This distribution is unsurprising as we deliberately chose the presence of industrial experience as an inclusion criteria in our list (see Sect. 4 on the previous page).

Second, the affiliations of the authors of the selected papers are in seven different countries: Ireland, USA, United Kingdom, Germany, Austria, Switzerland, Sweden, Columbia, and Italy. This distribution suggests a diversity across geographic zones.

5.1 Software Architecture Characteristics and the Context in Which They Were Deemed Important (RQ1)

Our analysis of the 13 included papers resulted in a list of 17 software architecture characteristics that were important to the implementation of DevOps/CD according to the experiences of the authors in these papers [16]. These are listed in the second column of Table 1. In the third column, we present the references to those papers addressing each characteristic. The number of references clearly indicates those software architecture characteristics which have been treated most frequently in relation to CD/DevOps in scientific literature. These are: deployability (CH2), testability (CH11), automation (CH3), loosely coupled (CH6), modifiability (CH1).

The characteristics in Table 1 have been described in more detail as part of *stage 1* of our research, which has already been presented in the ICSOFT 2019 conference [16]. Here we relate these characteristics to the context in which the authors of the 13 selected papers experienced them as important. For this purpose, we looked at the type of industrial projects in which the reported experience happened and observations on the characteristics of software architecture were collected. We found that half of the papers reported the context of very large organizations, for example, Deutsche Bank [3] – a leading German bank, Fujitsu [6] – a global IT consultancy, Ericsson [9] – a large telecommunication company, plus a large Swedish automotive company [11], and some large software process consultancy firms [1, 4].

We also looked at the architectural styles that matched these contexts. Eight out of our 13 selected papers indicated that the architectural style fitting DevOps/CD implementation is the one of micro-services. Micro-services are a set of small services that can be developed, tested, deployed, scaled, operated and upgraded independently, allowing organizations to gain agility, reduce complexity and scale their applications in the cloud in a more efficient way. Besides that, micro-services are very popular, they are being used and promoted by industry leaders such as Amazon, Netflix and LinkedIn [7]. Shahin et al. describe micro-services as the first architectural style to be preferred for CD practice, by designing fine-grained applications as a set of small services [5]. Three papers [9–11] state explicitly some specific benefits of employing the micro-services architecture concept. Micro-services are said to be helpful in increasing modularity and isolating changes and as a consequence increasing deployment frequency [13]. The experience report by Berger et al. [11], where the authors implemented CD practices in a team developing software for self-driving cars, reported how a loosely coupled micro-service architecture helped them move towards CD. Chen et al. argue that micro-service architectures feature many of the CD/DevOps enabling characteristics (CH2, CH7, CH8) and are (in combination with DevOps) the "key to success" of large-scale platforms [12].

Three other papers [5, 6, 8] explicitly state some downsides of the micro-services architecture. E.g. tracing errors and finding root causes of production issues traveling through multiple system components [8], resulting in increasingly complex monitoring (IS10) and logging (IS9) [5]. Plus, at the inception stage of a project a micro-services architecture might be less productive due to the required effort for creating the separate services and the necessary changes in the organizational structure, eventually as the project matures the efficiency of the micro-services architecture surpasses that of the monolithic architecture though [6].

Other authors [7, 8, 10] treat the suitability of the concept of micro-services in a particular context. Pahl et al. [10] state that the idea of micro-services has been discussed as a suitable candidate for flexible service-based system composition in the cloud in the context of deployment and management automation.

Furthermore, Schermann et al. [8] look at micro-services from a continuous experimentation perspective which is based on CD. These authors state that "continuous experimentation is especially enabled by architectures that foster independently deployable services, such as micro-services-based architectures".

Micro-services emerged as a lightweight subset of the Service-Oriented Architecture (SOA), it avoids the problems of monolithic applications by taking advantage of some of the SOA benefits [7]. Pahl et al. [10] note that loose coupling, modularity, layering, and composability are guiding principles of service-oriented architectures.

The last architectural style is vertical layering. It is mentioned by Shahin et al. [5] and refers to removing team dependencies by splitting software components into vertical layers (instead of horizontal layers, e.g. presentation, business and persistence). It can be argued if this is an architectural style on its own, as it is also a characteristic of micro-services and SOAs in general.

Table 1. Software architecture characteristics supporting CD/DevOps [16].

ID	Software architecture characteristics	Reference
CH1	Agility/Modifiability	[1, 2, 12]
CH2	Deployability	[2, 5, 12, 13]
CH3	Automation	[11–13]
CH4	Traceability	[11, 13]
CH5	Stateless components	[11]
CH6	Loosely coupled	[1, 10, 11]
CH7	Production versioning	[12]
CH8	Rollback	[12]
CH9	Availability	[12]
CH10	Performance	[12]
CH11	Testability	[2, 5]
CH12	Security	[2]
CH13	Loggability	[2, 5]
CH14	Monitorability	[2]
CH15	Modularity	[5, 10]
CH16	Virtualization	[10]
CH17	Less reusability	[5]

5.2 Application Domains (RQ2)

The experiences in our set of selected papers reported observations from a variety of domains, namely: banking, automotive, telecommunication, bookmaking, software and IT consulting. The example of a case study from a leading German bank [3] is representative for a major trend happening in the banking sector, namely the embarking on the concept 'Banking-as-a-Service' (BaaS). Transitioning to BaaS helps big banks re-invent themselves as assemblers of financial management solutions, tailored to meet specific customer needs. To succeed in this transition, banks increasingly more "componentize" their business architecture and underlying solution architectures of the systems they operate. The software architecture style they consider important to their future is micro-services [28] as it allows financial institutions to layer their technology offerings like building blocks rather than monolithic "systemware".

Furthermore, the experience described by Berger et al. [11], reflects a recent trend in the automotive sector adopting micro-services architectures. For example, car makers including Ford, Mercedes-Benz, and VW are actively adapting microservices/container architecture principles in developing Internet-of-Things enabled apps for their vehicles. Traditionally, in this sector, most automotive software architectures can be considered component based; in many cases, these components are however so tightly interconnected that the architectures should be considered monolithic. Companies realized that these monolithic architectures are will become a burden in the future and many embark to micro-service architecture to secure flexibility in the future.

Next, the experience reported by Chen [2] is about more than 20 CD projects observed PaddyPower, an Irish a multi-billion euro betting and gaming company, operating Ireland's largest telephone betting service. The author derived lessons learned on the roadblocks to CD and emphasized the role of micro-services in countering the effects of these roadblocks.

The report of Stahl and Bosch [9] focuses on the context of large network operators, many of which are transitioning to DevOps/CD (e.g. Ericsson, Swisscom). These authors report on their proposal for a continuous integration and delivery architecture framework and its large-scale empirical evaluation at Ericsson.

Finally, a number of papers address the specific context of software process improvement consultancy and IT firms (e.g. Endava [4], Fujitsu [6], Amazon Web Services [7]). For example, Woods (Endava [4]) puts forward the use of a number of architecture artefacts that one can re-thing for use in DevOps/CD contexts: release models, configuration management models, administrative models and support models. The approach that employs such models can be considered an architecture approach in itself.

5.3 Research Methods Being Used (RQ3)

Regarding the application of research methods in the publications that formed our set for analysis in this SLR, we found that only four out of the 13 papers explicitly stated the methodological origins of their selected method (see Table 2 below).

Table 2. Use of research methods in the 13 selected papers.

Ref.	Explicitness of research process	Research method	Experience report paper
[1]	Implicit	Case study	Yes
[2]	Implicit	Case study	Yes
[3]	Implicit	Case study	n/a
[4]	Implicit	Case study	Yes
[5]	Explicit	Interview-based study	n/a
[6]	Implicit	Case study	n/a
[7]	Explicit	Case study	n/a
[8]	Explicit	Mixed method: survey + interviews	n/a
[9]	Explicit	Mixed method: SLR + interviews + group workshops	n/a
[10]	Implicit	Case study	n/a
[11]	Implicit	Case study	n/a
[12]	Implicit	Case study	Yes
[13]	Implicit	Case study	n/a

These four articles [5, 7–9] leveraged the qualitative interview techniques for the purpose of their investigation. In two papers, the qualitative interviews formed a part of a mixed-method process, e.g. Stahl and Bosch [9] complemented the interviews with

group workshops, while Schermann et al. [8] used a survey together with interview. The remaining nine papers in the set of 13 only tacitly assumed the use of a case study. In fact, the authors provided rich details about the context of their organizations; there are descriptions either of project cases (e.g. [12]) or of case organizations (e.g. [2–4]). Moreover, we found four papers in the category of "experience reports"; this type of papers report on the application of a concept, method, or framework in one or several interesting industrial contexts, including the lessons-learned.

Regarding the ways in which the papers approach validity threats, we observe that threats have been explicitly discussed only by those authors that explicitly documented their research process.

6 Reflection on the Results

This section provides our reflection on our findings. First, we found 17 software architecture characteristics and as we could see from the findings regarding RQ2, these characteristics were deemed important in the context of large organizations transitioning to CD/DevOps. One can assume that these organizations maintain a large number of systems (some of which legacy systems) that are monolithic in nature. In an application landscape of monolithic systems it is then unsurprising that modifiability and agility are the most desired architecture characteristics. It is also not surprising that our SLR indicated the micro-services architecture style as the style considered the most suitable for this context.

Second, we found that the experiences published cover a broad range of application domains and companies operating in diverse business sectors. Also, from a broad range of countries located in Asia, America and Europe. This in itself has a positive implication: it allows us to think that the observations shared by the authors of the 13 papers are generalizable across application domains, business sectors and geographic zones.

Third, the finding that the case study approach was the one being used by most authors (see Table 2) matches the intuitive assumption that case studies are best in studied situations where the phenomenon of interest can be analyzed only in its real-world context (and can not be re-created in academic lab settings). However, many of the authors only implicitly mentioned the research method employed, which is at odds with the good practices and guidelines for reporting empirical software engineering research. This observation could be partly explained by the fact that many of the papers were published in practitioners' venues, such as the IEEE Software magazine, or on the practitioners' tracks of international scientific conferences. In both types of venues much more importance is placed on lessons learned and utility of the lessons learned for organizations than on the elaborate descriptions of the research method used.

7 Reflection on Bias in This SLR

In carrying out a SLR, it is also important to reflect on the criticality of researchers' own pre-knowledge and actions in reducing bias. As Archer et al. [22] state, knowledge in a scientific field is generated, developed, and interpreted by humans and relies on the

methods employed for this purpose. We reflect on four types of bias that are critical for SLRs: sampling bias, selection bias, and within-study bias, as described by Felson [23], plus expectancy bias as described by Cooper [24].

Sampling bias (including retrieval bias and publication bias) is concerned with the failure to capture all relevant studies' findings on the aspects of interest [23]. Retrieval bias refers to the risk that the key words in our search string might not be chosen well enough, which in turn means that a number of highly relevant papers would not be hit while executing the search string. We countered this risk by implementing the guidelines of Kuhrman et al. [20] in regard to this issue. In fact, we experimented with a variety of search strings and compared their results in Scopus. Next, publication bias is concerned with to the tendency of conferences and journals to publish empirical research results that challenge or change existing knowledge, while studies that confirm previous results are less frequently published [25]. This issue is apparent in new and emerging areas of the software engineering discipline, where published research commonly seeks to be original through proposing new definitions or developing new approaches (e.g. CD and DevOps), hardly ever replicating previous studies. To counter this challenge, some methodologists (e.g. Tranfield et al. [26]), recommend researchers consider both published and unpublished studies. However, in our protocol, we decided to use peer-reviewed literature only, which means grey literature was not included. Moreover, in our report on *stage 1* of our research [16], we noted that we compared our already reported findings (in [16]) against themes discussed in practitioners' online venues and we found no discussion theme that contradicts our findings.

Second, selection bias is concerned with the inaccurate design or application of the inclusion/exclusion criteria. To counter this bias, we followed the guidelines of Kuhrman et al. [20] in regard to the design of our criteria. We note also that this bias can be caused in situations in which a researcher is the author of a paper on the topic of the SLR. However, in case of this review, none of the authors has a publication on the topic of software architecture and CD/DevOps.

Third, within-study bias is concerned with the risk of variability in the data extraction procedures used by both researchers. We think however that this risk is minimal because the data extraction was simple and based on a form in which all the information was recorded in a systematic way.

Finally, expectancy bias is concerned with the synthesis of the information of the 13 primary study in this SLR. One reason for the occurrence of this bias is that researchers may have differing perspectives that influence the interpretation of study findings. During study synthesis, researchers may also be biased in seeking information that conforms to their expectations and may overlook or disregard perplexing information [24]. We countered this bias by having both researchers analyze all 13 papers in our set. This was possible because the number of papers was small. Each researcher reviewed each paper individually. After this, the researchers compared their analytical results. No disagreements happened in this process.

8 Related Systematic Literature Reviews

There are five literature studies that are related to our work. First, the 2014 study of Erich et al. [29] treats the question of how DevOps influences the performance of information system development and information system operation. This study looked at the evidence indicating specific benefits of DevOps. Unlike the SLR of Erich et al. [29], our research focused solely on the relationship between software architecture and DevOps.

The second SLR (2016) is on the practices of DevOps [33]. The authors of this SLR look into DevOps definitions and practices, while comparing DevOps with other software development method. This work explicitly states the practice of designing architecture as one belonging to DevOps.

The third review is the systematic mapping study of Rodrigues et al. [14] on the phenomenon of CD. These authors found that "CD demands a software architecture in which the product and its underlying infrastructure continuously evolve and adapt to changing requirements" (p. 15, [14]). This means that the underlying architecture should be flexible so that it can accommodate rapid feedback. This, in tur, points to architecture style that is modular and loosely coupled. Our findings agree with the findings of these authors. Our results however complement the findings in [14] by adding a list of architecture characteristics which are not among those mentioned in this mapping study [14].

The fourth study is the 2019 SLR of Céspedes et al. on the effects of DevOps on software product quality [31]. This SLR revealed a strong effect of the adoption of DevOps practices on reliability and maintainability of software products. The practices associated with DevOps, such as the minimum viable product, deployment automation, test automation, cloud computing and team cooperation, show a relationship with the improvement in software product quality. Our list of characteristics (Table 2) partly overlaps with those in this study [31]. In fact, modifiability (CH1 in Table 1) is a dimension of maintainability of a software architecture [32].

The fifth review is the mapping study of Di Francesco et al. [30] on the phenomenon of architecting with micro-services. As part of these authors' analysis, the study yielded a list of architecture quality attributes that were treated in studies on micro-service architecture. Performance, maintainability, and functional suitability were found as the most studied attributes. Although these attributes were the focus of researchers working on micro-services, our SLR did not find them as the most frequently mentioned architecture characteristics from DevOps/CD perspective (see Table 1). This difference could be explained with the fact that many researchers investigated the architecting-with-micro-cervices practice in contexts in which large organizations transition from monolithic to micro-services architecture without necessarily employing DevOps and CD.

9 Summary and Implications

Using 13 publications on software architecture in DevOps/CD, this SLR indicates that:

(1) there are 17 software architecture characteristics which are beneficial for CD and DevOps adoption according, according to published literature;
(2) micro-services are recommended architectural style in this context, and
(3) large-scale organizational contexts are the most studied, and
(4) qualitative approaches (case study based) are the most applied research method.

Our review has some implications for researchers. As we found that most knowledge comes from large organizational settings in which there are many systems with monolithic architecture, it may be interesting for researchers to focus on greenfield projects in DevOps/CD. What are the architecture styles that DevOps/CD teams adopt in case of developing new systems that did not exist before? This seems a worthwhile topic for exploratory research in the future.

Another question related to the context of start-ups. While DevOps seems a logical choice for many startups in the IT and software marketplace, practitioners warn (e.g. in [27]) that the micro-service architecture may not always be the best choice in the startup context. Understanding how startup companies embracing DevOps choose their architecture styles is an industry-relevant line for research in the future.

Finally, research on the topic of interest in this SLR so far has been qualitative in nature taking into account the real-world contexts in which architecture characteristics were deemed most beneficial for DevOps/CD. However, only four of our 13 analyzed papers were explicit on the research method used. This poses a threat to the validity of the reported lessons learned by practitioners and possible candidates for good practices. From research and knowledge generation perspective, these lessons learned and candidate good practices could serve as lists of hypotheses that researchers could test in other settings in order to generate empirical evidence to draw more specific conclusions. Only then, we could make some well-substantiated claims about the software architecture characteristics beneficial for the implementation of DevOps/CD.

Our SLR has some practical implications. First, it brings good news to large organizations regarding the fit of the micro-service architecture style to DevOps/CD. Our included papers presented working examples of a broad range of industry sectors and countries, which allows us to conclude the viability of the micro-service architecture as an option to consider.

Second, software architects who may be interested in developing some architecture guidelines in their organizations using DevOps/CD might consider our list of characteristics as one possible starting point along with other considerations, such as current IT project portfolio and proportion of green-field projects in it.

References

1. Sturtevant, D.: Modular architectures make you agile in the long run. IEEE Softw. **35**(1), 104–108 (2017)

2. Chen, L.P.: Towards architecting for continuous delivery. In: Bass, L., Lago, P., Kruchten, P. (eds.) 12th Working IEEE/IFIP Conference on Software Architecture, pp. 131–134 (2015)
3. Erder, M., Pureur, P.: Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World, pp. 1–303. Morgan Kaufmann, Burlington (2015)
4. Woods, E.: Operational: The forgotten architectural view. IEEE Softw. **33**(3), 20–23 (2016)
5. Shahin, M., Babar, M.A., Zhu, L.: The intersection of continuous deployment and architecting process: Practitioners' perspectives (2016)
6. Elberzhager, F., Arif, T., Naab, M., Süß, I., Koban, S.: From agile development to DevOps: Going towards faster releases at high quality – experiences from an industrial context. In: Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2017. LNBIP, vol. 269, pp. 33–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49421-0_3
7. Villamizar, M., et al.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 10th Computing Colombian Conference (10CCC) (2015)
8. Schermann, G., et al.: We're doing it live: a multi-method empirical study on continuous experimentation. Inf. Softw. Technol. **99**(7), 41–57 (2018)
9. Ståhl, D., Bosch, J.: Cinders: The continuous integration and delivery architecture framework. Inf. Softw. Technol. **83**(3), 76–93 (2017)
10. Pahl, C., Jamshidi, P., Zimmermann, O.: Architectural principles for cloud software. ACM Trans. Internet Technol. **18**(2), 1–23 (2018)
11. Berger, C., et al.: Containerized development and microservices for self-driving vehicles: Experiences & best practices. In: 2017 IEEE International Conference on Software Architecture Workshops, pp. 7–12 (2017)
12. Chen, H.M., et al.: Architectural support for DevOps in a neo-metropolis BDaaS platform. In: 2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop, pp. 25–30 (2015)
13. Bass, L.: The software architect and DevOps. IEEE Softw. **35**(1), 8–10 (2017)
14. Rodríguez, P., et al.: Continuous deployment of software intensive products and services: a systematic mapping study. J. Syst. Softw. **123**, 263–291 (2017)
15. Bass, L., Weber, I., Zhu, L.: DevOps: A Software Architect's Perspective. Addison-Wesley, Boston (2015)
16. Bolscher, R., Daneva, M.: Designing software architecture to support continuous delivery and DevOps: a systematic literature review. In: ICSOFT 2019, pp. 27–39 (2019)
17. Humble, J., Farley, D.: Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson Education, London (2010)
18. Wieringa, R., Daneva, M.: Six strategies for generalizing software engineering theories. Sci. Comput. Program. **101**, 136–152 (2015)
19. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. Keele University, UK (2007)
20. Kuhrmann, M., Méndez Fernández, D., Daneva, M.: On the pragmatic design of literature studies in software engineering: An experience-based guideline. Emp. Softw. Eng. **22**(6), 2852–2891 (2017)
21. Pawson, R.: The Promise of a Realist Synthesis, Working Paper No.4, ESRC Evidence Network, Centre for Evidence Based Policy and Practice (2001). http://www.evidencenetwork.org/Documents/wp4.pdf
22. Archer, M., Bhaskar, R., Collier, A., Lawson, T., Norrie, A.: Critical Realism: Essential Readings. Routledge, London (1998)
23. Felson, D.T.: Bias in meta-analytic research. J. Clin. Epidemiol. **45**(8), 885–892 (1992)
24. Cooper, D.H.M.: Research Synthesis and Meta-Analysis: A Step-by-Step Approach. Sage Publications Inc., Los Angeles (2010)

25. Littell, J.H., Corcoran, J., Pillai, V.: Systematic Reviews and Meta-Analysis. Oxford University Press, Oxford (2008)
26. Tranfield, D., Denyer, D., Smart, P.: Towards a methodology for developing evidence-informed management knowledge by means of systematic review. Br. J. Manag. **14**(3), 207–222 (2003)
27. https://adevait.com/software/why-most-startups-dont-need-microservices-yet
28. Bucchiarone, A., et al.: From monolithic to microservices: an experience report from the banking domain. IEEE Softw. **35**(3), 50–55 (2018)
29. Erich, F., Amrit, C., Daneva, M.: A mapping study on cooperation between information system development and operations. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 277–280. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13835-0_21
30. Di Francesco, P., Lago, P., Malavolta, I.: Architecting with microservices: a systematic mapping study. J. Syst. Softw. **150**, 77–97 (2019)
31. Céspedes, D., Angeleri, P., Melendez, K., Dávila, A.: Software product quality in DevOps contexts: A systematic literature review. In: Mejia, J., Muñoz, M., Rocha, Á., Calvo-Manzano, J.A. (eds.) CIMPS 2019. AISC, vol. 1071, pp. 51–64. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-33547-2_5
32. Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H.: Architecture-level modifiability analysis (ALMA). J. Syst. Softw. **69**(1–2), 129–147 (2004)
33. Jabbari, R., Bin Ali, N., Petersen, K., Tanveer, B.: What is DevOps?: a systematic mapping study on definitions and practices. In: XP Workshops 2016, p. 12 (2016)