Adopting Continuous Delivery and Deployment: Impacts on Team Structures, Collaboration and Responsibilities

Mojtaba Shahin [a], Mansooreh Zahedi [b], Muhammad Ali Babar [a], Liming Zhu [c]
[a] CREST - The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia
[b] CREST - The Centre for Research on Engineering Software Technologies, IT University of Copenhagen, Denmark
[c] Data61, Commonwealth Scientific and Industrial Research Organisation, Sydney, NSW 2015, Australia
mojtaba.shahin@adelaide.edu.au, mzah@itu.dk, ali.babar@adelaide.edu.au, liming.zhu@data61.csiro.au

ABSTRACT

Context: Continuous Delivery and Deployment (CD) practices aim to deliver software features more frequently and reliably. While some efforts have been made to study different aspects of CD practices, a little empirical work has been reported on the impact of CD on *team structures*, *collaboration* and *team members' responsibilities*. **Goal**: Our goal is to empirically investigate how Development (Dev) and Operations (Ops) teams are organized in software industry for adopting CD practices. Furthermore, we explore the potential impact of practicing CD on collaboration and team members' responsibilities. **Method**: We conducted a mixed-method empirical study, which collected data from 21 in-depth, semi-structured interviews in 19 organizations and a survey with 93 software practitioners. **Results**: There are four common types of team structures (i.e., (1) *separate Dev and Ops teams with higher collaboration*; (2) *separate Dev and Ops teams with facilitator(s) in the middle*; (3) *small Ops team with more responsibilities for Dev team*; (4) *no visible Ops team*) for organizing Dev and Ops teams to effectively initiate and adopt CD practices. Our study also provides insights into how software organizations actually improve collaboration among teams and team members for practicing CD. Furthermore, we highlight new responsibilities and skills (e.g., monitoring and logging skills), which are needed in this regard.

CCS Concepts

• Software and its engineering → Software development process management

KEYWORDS

Continuous delivery and deployment; development and operations teams; collaboration; empirical study;

ACM Reference format:

M. Shahin, M. Zahedi, M. Ali Babar, and L. Zhu. 2017. Adopting Continuous Delivery and Deployment: Impacts on Team Structures, Collaboration and Responsibilities. In *Proceedings of 21st International Conference on Evaluation and Assessment in Software Engineering, Karlskrona, Sweden, June 15-16, 2017 (EASE'17)*, 10 pages.
DOI: 10.1145/3084226.3084263

1. INTRODUCTION

Development and Operations (DevOps) has been embraced by software development industry as a new paradigm to improve collaboration, communication, and integration between software development and operations teams [5]. Continuous Delivery and Deployment (CD) practices are DevOps practices enabling IT organizations to frequently and sustainably release software into production environments [4; 5]. A growing number of indicators make it clear that CD practices are increasingly making inroads in industrial practices across various domains and sizes of organizations. For example, a number of highly innovative companies such as Facebook, Netflix, and Etsy have adopted these practices to deliver value to their customers quicker [40]. In the meanwhile, the highly complex and challenging nature of DevOps practices, particularly CD practices, make it inevitable that organizations improve their skills, form the right teams, and investigate organizational processes, practices, and tool support to gain anticipated benefits from DevOps practices [8]. With the increasing popularity of CD practices, research community has been conducting extensive research efforts to understand how organizations initiate and implement these practices. For example, a few papers have investigated the challenges that organizations may face in adopting CD practices [8; 27; 35]. The other area of interest in CD is to provide and integrate appropriate technologies and tools to support automated configuration and deployment processes [46]. It is asserted that achieving CD may require a new way of working and changes in team structures and responsibilities [2; 3; 8]. Furthermore, CD practices demand tighter and stronger collaboration and integration among teams and team members [45]. However, there is no systematic research about how organizations actually form and arrange Development (Dev) and Operations (Ops) teams and also how they increase collaboration among teams and team members to optimally embrace CD practices. We assert that such questions should be explored and answered through empirical studies involving practitioners from diverse organizations rather than through one case company or one practitioner's perspective [2; 3]. To address this gap, we report an empirical investigation to address the following research questions:

RQ1. *How are Dev and Ops teams organized to initiate and adopt continuous delivery and deployment?*

RQ2. *How is collaboration among teams and team members improved for adopting continuous delivery and deployment?*

RQ3. *How does adoption of continuous deliver and deployment impact on team members' responsibility?*

To answer these research questions, we used a mixed-method study consisting of interviews and survey. We conducted 21 semi-structured interviews with software practitioners from 19 organizations. We assessed and quantified the findings from the interviews using a survey that obtained responses from 93 practitioners. The main findings from our research are:

(i) There are four common types of patterns for organizing Dev and Ops teams to effectively initiate and adopt CD practices: (1) *separate Dev and Ops teams with higher collaboration*; (2) *separate Dev and Ops teams with facilitator(s) in the middle*; (3) *small Ops*

EASE'17, June 15-16, 2017, Karlskrona, Sweden
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4804-1/17/06$15.00
DOI:10.1145/3084226.3084263.

team with more responsibilities for Dev team; (4) *no visible Ops teams*.

(ii) The participants shared that *co-locating teams*, *rapid feedback*, *joint work and shared responsibility, using collaboration tools more often, increased awareness and transparency,* and *empowering and engaging operations personnel* enabled them to increase the collaboration among teams and team members in the path of adopting CD.

(iii) Team members have three key high level changes in their responsibilities: *expanding skill-set*, *adopting new solutions aligned with CD*, and *prioritizing tasks*.

The paper is structured as follows: Section 2 provides the background and related work. Section 3 describes research methodology. We report our findings in Section 4. Section 5 discusses the threats to the validity of the results. Finally, Section 6 closes the paper with discussion and conclusions.

2. BACKGROUND AND RELATED WORK

2.1 Background

DevOps aims at reducing the time between committing a change and deploying the change into production without quality degradation [5]. A set of practices are associated with DevOps including Continuous Delivery and Deployment (CD), automated testing, and infrastructure as code [4; 5]. Continuous delivery and deployment practices are highly correlated and intertwined, in which the precise definitions of these practices are often missing [13; 15; 20]. It is sometimes hard to differentiate these practices from each other and their meanings heavily depend on how a given organization employs them [28; 43]. Continuous delivery ensures that an application is at a releasable state at any time [44]. Continuous deployment extends continuous delivery by automatically and steadily deploying an application to production as long as automated tests and checks are passed. Whilst in continuous delivery practice, the management decides when changes should be delivered to customers, continuous deployment has no manual steps or decisions; as soon as developers commit a change, the change is deployed to production through a continuous deployment pipeline [11]. Applicability of these two practices is another source of difference. Whilst all types of systems and organizations *can* practice continuous delivery, continuous deployment may not be suitable to all types of organizations or systems [20; 33; 39]. Since these practices are intertwined and often used interchangeably, we refer Continuous Delivery and Deployment (CD) as CD practices in this paper.

2.2 Related Work

There are a number of empirical studies that have investigated the challenges and practices of adopting DevOps and CD [8; 27; 30; 41]. Among the reported challenges (e.g., monolithic architectures [41]) and practices (e.g., management support [40]), the studies have also briefly discussed the skills required for practicing CD, and how collaboration and coordination among teams and their members can be consolidated for this purpose.

Savor et al. [40] report that the developers needed to gain new skills as a result of implementing continuous deployment at Facebook and OANDA. The studied companies assigned new developers to the release engineering team for several months. Claps et al. [8] identified 20 technical and social challenges (e.g., team experience and team coordination) that a single case company faced in transition towards continuous deployment. In order to move from continuous integration (CI) to continuous deployment, the case company studied in [8] leveraged CI developers' experience by integrating automated continuous deployment of software into the existing CI workflow of developers. This approach helped them to reduce the learning curve for developers. Other studies discuss that when a project adopts CD practices, it would be helpful to define new roles and teams in software development lifecycle to smooth this path. Krusche and Alperowitz [26] define hierarchical roles such as release manager and release coordinator to adopt and implement continuous delivery in multi-customer projects. It is argued that these roles improve coordination among team members. Other studies argue that establishing a dedicated team for design and maintenance of infrastructure and deployment pipeline helps organizations to smoothly transform to CD and reduce release cycle time [7; 40].

Wettinger et al. [45] present the idea of solution repositories to provide efficient collaboration between developers and other team members (i.e., operations stakeholders). Each team in software development lifecycle may use their own solutions and repositories to build and maintain knowledge and documents around corresponding solutions. This approach could significantly hinder knowledge sharing and collaborative work in a team. The collaborative solution repositories automatically collect and store different solutions and their metadata from diverse environments (e.g., test environments) and sources (e.g., Chef). Then this data is utilized to establish consolidated knowledge base instances for supporting collaboratively work.

Nybom et al. [36] conducted a case study with 14 practitioners in an organization to investigate the potential impact of mixing responsibilities between developers and operations staff. The study reveals that mixing responsibilities, among other impacts, remarkably increases collaboration and trust, and fosters team members' workflow. However, this approach is associated with a number of negative implications. For example, given more responsibilities to developers and constantly learning about operations tasks might demotivate some developers to have collaboration with operations personnel.

França et al. [16] conducted a multivocal literature review to characterize DevOps principles and practices. The study highlights that developers and operations personnel need to gain both social (e.g., communication) and technical (e.g., math skills for performance analysis) skills to truly perform DevOps. This also enables team members to effectively collaborate on fixing bugs. The study also found a number of practices to improve collaboration among team members including role rotation, face-to-face communication, and open information.

It should be noted that none of the above-mentioned studies has systematically and empirically explored the actual impact of CD practices on the structure of Dev and Ops teams, team members' responsibilities, and collaboration among them. Whilst analyzing our data for RQ1, we came across a few blogs [2; 3], which suggest different team structures (e.g., Ops as Infrastructure-as-a-Service) for DevOps success. That increased our confidence in the importance of exploring how Dev and Ops teams are organized in practice for adopting CD. Our study is based on qualitative findings from 21 in-depth interviews that are assessed and quantified by a survey with 93 software practitioners from diverse organizations. We assert that our findings provide an evidence-based and detailed view of different team setups when adopting CD, as they are not restricted to a single case company or observations of one practitioner.

3. METHODOLOGY

We have adopted a mixed-method empirical study to answer our research questions. Mixed-method study makes use of both qualitative and quantitative research methods for data collection and analysis [12] in order to enable researchers to compensate weaknesses of both methods [9]. We applied the mixed-method study with *sequential exploratory strategy* [12], i.e., characterized by collecting and analyzing qualitative data, followed by quantitative data to assist interpretation of qualitative findings. We have collected qualitative data through 21 in-depth, semi-structured interviews [12; 24]. Then we carried out a survey of 93 practitioners to further gain the evidence and understanding of our findings from the interview study. For both studies, we developed and followed research protocol using appropriate guidelines [19; 25].

3.1 Interviews

Protocol: We conducted in-depth, semi-structured interviews with 21 software practitioners from 19 organizations. The relevant parts of the interviews for this study consisted of 10 open-ended questions[1] and aimed at capturing experiences and reflections of participants on how adopting CD influenced their team structure, responsibilities and collaboration model. Due to geographical distribution of participants, it was not feasible to perform face-to-face interviews. Hence, we used Skype for majority of the interviews, and only two interviews took place through email. We shared the interview guide with the participants before conducting the interviews in order to help the participants to be prepared for answering the questions and engaging in discussions [19]. During the data collection, some of the questions got improved based on the feedback we gathered from initial interviews. We audio-recorded and transcribed all the interviews for reliable and in-depth analysis.

Participants*:* The selection of the interviewees was based on purposive sampling method [37]. Given the nature of our research questions, we only looked for potential interviewees who either had worked for organizations adopting DevOps/CD practices or were involved in DevOps/CD consulting organizations. The interviewees were found through different channels such as our personal network, exploring the lists of speakers and attendees of relevant industrial conferences. Having rigorously analyzed their profiles to ensure fulfilling our study criteria, we sent them invitation email to participate in our study and offered a free DevOps book (i.e., "*DevOps: A Software Architect's Perspective*" [5]) as a gift for inspiration. We chose the interviewees from a variety of professional and industrial backgrounds (e.g., different levels of work experiences and various project roles) and their organizations differed from each other in terms of domain and size. Utilizing "snowballing technique" [17], we also asked the interviewees to indicate the potential participants.

Analysis*:* For the sake of refer-ability and reliability, we opted for digitalized analysis using NVivo[2] software (i.e., a qualitative data analysis tool). We used thematic analysis technique applied in software engineering for analyzing data [10]. We initiated the analysis by breaking down the transcripts into three high-level segments according to our research questions: the impact of practicing CD on team structures (RQ1), collaboration (RQ2) and team members' responsibilities (RQ3). Then we rigorously reviewed the transcripts, and extracted and coded data related to each of the research questions. Initial codes were further combined to construct potential themes. In the next step of the analysis, we constantly reviewed and compared the extracted themes against each other to identify the themes that needed to be merged or excluded (e.g., due to insufficient evidence). The themes were further grouped into higher-order categories. Finally, we verified the trustworthiness of the core themes and gave them a precise title. Figure 1 partially demonstrates an example of our analysis that led to the core theme of "*No visible Ops team*".

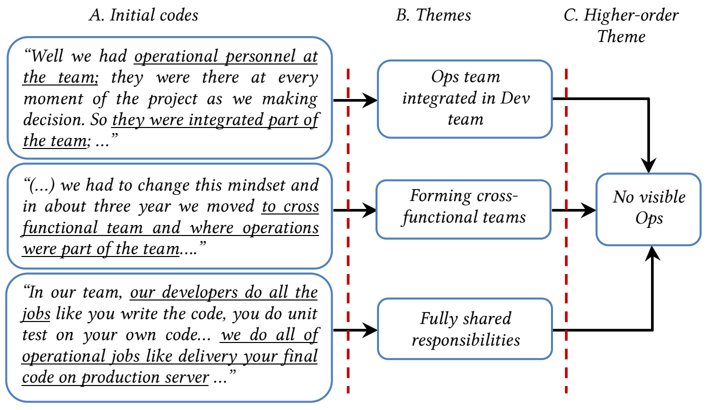
A. Initial codes
B. Themes
C. Higher-order Theme
"Well we had operational personnel at the team; they were there at every moment of the project as we making decision. So they were integrated part of the team; ..."
"(...) we had to change this mindset and in about three year we moved to cross functional team and where operations were part of the team...."
"In our team, our developers do all the jobs like you write the code, you do unit test on your own code... we do all of operational jobs like delivery your final code on production server ..."
Ops team integrated in Dev team
Forming cross-functional teams
Fully shared responsibilities
No visible Ops

Figure 1: The steps of applying thematic analysis on interview transcripts

3.2 Survey

Protocol: Our survey design followed the guidelines of Kitchenham and Pfleeger [25]. We aimed at assessing and quantifying the interviews' findings. Hence, the survey questions were designed based on the qualitative results that emerged from the interviews. The survey preamble included the objective of our study, participation requirements and brief definitions of continuous delivery and deployment practices for our research. Apart from demographic questions (e.g., role and experiences), we particularly asked about how practicing CD has influenced team structure, responsibilities and collaboration of team members. The relevant survey questions used for this study contained 11 questions[1] including demographic (4 questions), five-point Likert-scale (3 questions), single-choice (2 questions) and open-ended (2 questions). All the questions were marked as mandatory to collect as much complete data as possible. We also considered the "Other" field for the single-choice questions to capture participants' additional inputs [18]. We used Likert-scale questions for rating three types of statements: (i) agreement with a given statement (i.e., from *strongly agree* to *strongly disagree*), (ii) recognizing the importance of a given statement (i.e., from *very important* to *unimportant*), (iii) the likelihood of experiencing a given statement (i.e., from *not at all* to *very much*).

Participants: We used different techniques and channels to advertise the survey and collect responses. We started with posting the survey to several LinkedIn groups related to DevOps and CD. We also invited around 4000 GitHub users for participation in the survey through email. Though we incentivized practitioners to participate in our research by different means (e.g., offering DevOps book and sharing findings), the initial response rate was quite low (i.e., less than 10% of all the responses were received through these techniques). It was also experienced by previous researchers that advertising survey through social media might fail to attract a large number of participants [34]. Therefore, we applied the same process that we used to recruit the interviewees. We searched for practitioners on the web with the relevant experiences by exploring the topical forums (e.g., industry-driven conferences on

1 The interview and survey questions are available at https://mojtabashahin.files.wordpress.com/2016/02/questions.pdf

[2] http://www.qsrinternational.com/

DevOps), carefully reviewed the practitioners' profiles (e.g., speakers or attendees of the conferences). During this phase, we sent 487 invitations to highly relevant practitioners. Finally, we received 93 responses.

Analysis: We have analyzed the survey data using two techniques. For the responses collected on the Likert-scale and close-ended questions (e.g., single choice), we applied descriptive statistics [34]. For the open-ended questions that resulted in qualitative data (i.e., 1-2 lines of description), we applied the conceptualized thematic analysis method described in Section 3.1.

4. FINDINGS

We start reporting the findings by providing an overview of the participants and their respective organizations. We then present our findings about different team structures in software organizations to adopt CD, followed by strategies and practices adopted to effectively improve collaboration. Finally, we describe how CD adoption may change responsibilities of team members. Due to confidentiality purpose, the anonymity of the interviewees, the survey participants and their respective organizations has been strictly maintained when presenting the findings.

4.1 Participants Profiles

21 practitioners (i.e., indicated by **P1** to **P21**) from 19 organizations in 9 countries participated in the interviews. We also received 93 responses for the survey (i.e., indicated by **R1** to **R93**). Majority of the participants were architects (47), followed by consultants (14), DevOps engineers (11), and developers (8). The rest of them include software engineers (7), team leads (6), executives (e.g., CTO, 5), operations engineers (4), program managers (4), and others (13). 73.6% of the participants had more than 10 years of experience in software industry, 15.7% 6-10 years, 7% 3-5 years, and 3.5% 1-2 years. Both the interviewees and the survey respondents were fairly evenly distributed in large (>1000 staff), medium (100-1000 staff) and small (<100 staff) organizations, in which in total 46 participants worked in large, 37 in medium-sized and 31 in small organizations. Furthermore, the interviewees and the survey participants worked for very diverse organizations in terms of domains such as consulting and IT services (43), financial (12), e-commerce (10), and telecommunication (8).

4.2 Team Structures for Adopting CD Practices (RQ1)

This section reports how Development (Dev) and Operations (Ops) teams are organized to implement CD practices (See Figure 2). First, we present the main patterns of team structures for this purpose, which are extracted from the interviews. Second, we assess and quantify these patterns by indicating the number of the survey respondents reported these patterns.

Separate Dev and Ops teams with higher collaboration: Our analysis has disclosed that for a couple of the interviewees' organizations, in particular hierarchical ones, adopting CD does not necessarily mean huge changes in team structure or complete breakdown of silos (i.e., divisions of labor) between teams. They tried to leverage their existing Dev and Ops teams by providing the needed infrastructures and emphasizing the culture of empowerment in order to make a higher and tighter collaboration between Dev and Ops teams (See Figure 2-A). Through this strategy, they were able to achieve DevOps and CD goals as much closeness as they could. The amount of collaboration between teams and team members, in particular application developers and operations team, increased after adopting CD. It was explained by one of the interviews' participants in the following words:

"*They [organization] did very successful continuous delivery even though they have separate teams for development and operations. So I mean you are on spot that there should be close Ops cooperation but it is not necessarily [to have] the full DevOps in the sense of making the teams do operations and development tasks by themselves. I think you do need to have close collaboration but you do not need to have teams to do both Ops and Dev [tasks]*" ***P11***.

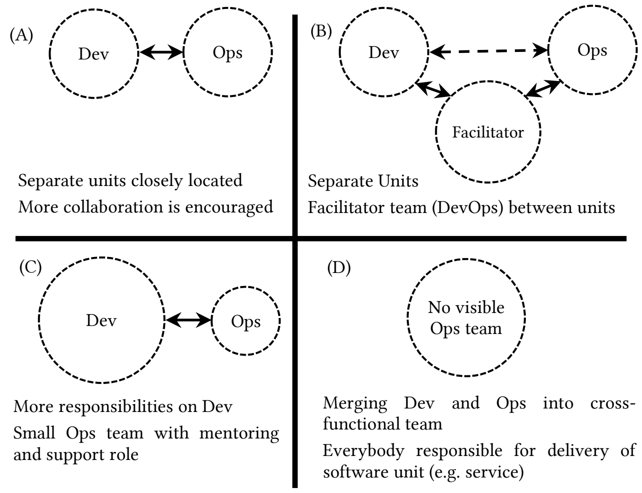
(A)
Dev
Ops
Separate units closely located
More collaboration is encouraged
(B)
Dev
Ops
Facilitator
Separate Units
Facilitator team (DevOps) between units
(C)
Dev
Ops
More responsibilities on Dev
Small Ops team with mentoring and support role
(D)
No visible Ops team
Merging Dev and Ops into cross-functional team
Everybody responsible for delivery of software unit (e.g. service)

Figure 2: Team Structure for effectively initiating CD practices

We found that placing operations team next to developers (e.g., in the same office) and encouraging them to have more collaboration and face-to-face communication with other team members are simple strategies adopted by the interviewees' organizations to bridge the collaboration gap between Dev and Ops teams. One interviewee (i.e., a program manager) pointed out this in these words:

"*There are two sub-groups, who reported me. There is some division of labor who focus on development and there is another subgroup of 3 people who are focusing on the operations and deployment. But they sit next to each other and they work very closely together*" ***P7***.

We asked the survey participants to determine whether this pattern describes the structure of Dev and Ops teams in their respective or client organizations. As shown in Table 1, of 93 survey responses to this question, 33 (35.4%) of the respondents indicated that they still have separate Dev and Ops teams; however, it was reported that collaboration and coordination among even the separate teams had significantly improved. Interestingly, this pattern was mainly adopted by large organizations, followed by medium-sized organizations as 48.6% of the large organizations had structured their Dev and Ops teams in this way. While, only 5 out of 26 small organizations chose this pattern.

Separate Dev and Ops teams with facilitator(s) in the middle: As part of the strategy to improve communication and collaboration between developers and operators, some interviewees' organizations would go a step further by defining and establishing a team, for example, so called DevOps team, to facilitate communication and collaboration between Dev and Ops teams (See Figure 2-B). This team acts as an integrator between these teams to consolidate work together and knowledge sharing. The participant **P4** highlighted the role of this team as the follows:

"*We had DevOps engineer [who has] job to integrate between development and operations to be the primary developer responsible for integrating between Dev and Ops to make sure the all changes are applicable to operations"* ***P4***.

17.2% of the survey respondents stated that they are using a facilitator team as enabler for communication and collaboration. Only one small size organization used this pattern, remainders were large (9) and medium-sized (6) organizations.

Small Ops team with more responsibilities for Dev team: DevOps often recommends that developers take more accountability about their code in production environments [5]. Some interviewees' organizations have gradually and smoothly shifted operational responsibilities from infrastructure and operations teams to Dev team. By applying this change, Ops team is more responsible for mentoring, coaching and helping developers to write operational aspects of code, for example writing provisioning code. This strategy enabled the interviewees' organizations to make operations process easier and helped developers to commit codes that made less trouble. This is mainly because Ops team influenced the way the applications were configured to make them easier to deploy. Furthermore, Ops team may still exist to handle initial incidents in production environments. Hence, development team is not available like 24/7 to address incidents in production and initial incidents handling will be out of developers' accountability. As one interviewee commented:

"*They (operations team) often would pass the problems to the development team, if they cannot solve the problem itself and then the development team will get involved in operational things, incidents, that kind of things*" ***P18***.

Organizations within this category still had a distinct operations team, albeit a small one with limited responsibilities (See Figure 2-C). According to the interviews' participants, operations team is still needed to support deployed system in the production. They are mainly in charge of running the system, monitoring it, and fixing the performance issues. As **P12** stated that:

"*The organization that I am talking about is very hierarchy organization and we are not able to inroad and change the organizational hierarchy. I really like these things [operations tasks] run as product team where you have Ops people embedded in product team and then whole team working together. We have not got that state but we could get through months and months like talking to each other and having bear with each other*" ***P12***.

We could not conclude that the Ops team in this category is a part of Dev team as there are always a bunch of tasks that are not really related to or out of expertise of a development team. 27 out of 93 survey respondents indicated that there is a very small Ops team (e.g., 2-3 people) in their organizations to do specific tasks and most of the responsibilities of Ops team have been shifted to the Dev teams. The distribution of this pattern was almost similar among large (8), medium-sized (9), and small organizations (10). There is always a need to be someone on duty, particularly in critical systems such as financial systems, which has to be available 24/7. An IT architect, who worked in a company specialized in DevOps and CD and helped other organizations to adopt DevOps practices, pointed out that:

"For me you may want to know, I have not seen in many organizations that DevOps team, the ideal situation, is really happened as a practice at the moment. So what I mean this is a full responsibility; they are really multi-disciplinary team and they can do all the technologies themselves and that requires high skilled people to learn real DevOps team" ***P9***.

No visible Ops team: Our analysis has revealed that in a few organizations, the Ops team has been an integrated part of Dev team (See Figure 2-D). There is no specific and visible Ops team; all team members have a shared responsibility and purpose to cover the entire spectrum of software application, from requirement gathering, to continuously deploying, monitoring, and optimizing application in production environments.

"*Well we had operational personnel at the team; they were there at every moment of the project as we were making decision. So they were integrated part of the team; there is no communication overhead for operation teams because we have no operations in a separate operations team"* ***P13***.

The results from the survey show that 17.2% of the respondents, especially in small organizations, stated that they do not have visible and distinct Ops team. Those organizations have structured team members in cross-functional team for each software unit (e.g., service and component); therefore, each team includes developers, business analyst, quality assurance (QA) people, and operations people. It is also asserted that creating cross-functional team (e.g., operations team is completely embedded in development team) necessitates highly skilled people and this pattern have usually been found in Start-up or highly innovative web companies [2; 3].

"*Initially we had separated operations team. There was a huge concern from the business, because people came from IT background, you had to have developers who were far away from the production; because it's risk stuff; we had to change this mindset and in about three years we moved to cross functional team and where operations were part of the team*" ***P14***.

A small number of the interviewees emphasized that if organizations want to efficiently adopt and implement DevOps practices, in particular CD practices, they cannot really have operations silo (i.e., separate Ops team), even small one. Having operations silo may lead to a lot of frictions in deployment process and fail organizations to achieve the real anticipated benefits of CD practices.

Five respondents chose the "Other" field, but they did not provide a new pattern for organizing Dev and Ops teams. They mainly used this field to describe their team structures in other ways. Thus, we assigned them to existing categories. For example, **R71** stated that "*We have 2 Web Ops teams serving > 30 Dev teams. The Web Ops provide infrastructure, tooling and deployment, but the Dev teams are monitoring and manage their own services in production*".

Table 1: Survey results on patterns of organizing Dev and Ops teams for initiating and adopting CD practices

Dev and Ops teams patterns	#	%
Separate Dev and Ops teams with higher collaboration	**33**	**35.4**
Separate Dev and Ops teams with facilitator(s) in middle	**16**	**17.2**
Small Ops team with more responsibilities for Dev team	**27**	**29**
No visible Ops team	**16**	**17.2**
Not Available	**1**	**2.2**

4.2.1 Team Structures for Designing Pipelines

It is asserted that the success of adopting DevOps practices (e.g., in particular CD) in organizations would heavily depend on the choice of appropriate tools, technologies, infrastructures, and level of automation to implement Continuous Deployment Pipeline (CDP) or also known as continuous delivery pipeline [38; 44]. It is worth noting that organizations used different terminologies to refer CDP. We observed that the participants' organizations adopted following models to introduce CDP:

Organization-driven model: Software organization may found a team to build and maintain platforms, infrastructures and tool chain (e.g., Jenkins and Chef) to set up a (semi-) automated CDP [1]. Then all project teams in the organization are able to use this CDP to build, test, package, and run their applications [5]. Having a common CDP enables organizations to improve consistency, governability and team productivity [6]. Among 19 interviewees' organizations, we observed three different patterns for organizing that team. For all cases, first an organization builds CDP by applying one of following patterns, and then multiple projects simultaneously are fed into and ran on established CDP.

Centralized Team: According to our participants, adopting and scaling CD practices at a large organization with multiple teams and applications necessitates a CDP that supports traceability, scalability and flexibility [42]. The CDP must be able to perform no matter how large or many applications it processes, or how large their test suites are. It must also be flexible in a way that organizations can extend and tune it or parts of it without major disruption or major effort. Furthermore, the CDP should support traceability, in which enables a wide range of stakeholders to understand what is happening, what has happened, and why. For some interviewees' organizations, that is achievable by establishing a dedicated and centered team to design, develop and continuously improve CDP in the long term. One of the interviewees told us: *"We had Squad that was responsible for basically taking care of the platform. ... So my colleagues, Squad was responsible for DevOps platform layer"* ***P6***.

This was the most commonly chosen pattern by the survey respondents, with 39.7% (37) choosing that a central team in their organizations designed a CDP that would work best for all teams and applications. We found that this model of forming CDP team mainly appeared and practiced in large (20) and medium-sized (12) organizations.

Temporary Team: In contrast to the previous pattern, CDP in this pattern is built by a temporarily established team in an organization and then the members of that temporary team join other teams because there is no need for them anymore. As one interviewee explained that:

"Once we have set up continuous integration, they would call pipeline, once the pipeline there, and if there is no problem, we will go back to the pool, we don't stay all days." ***P8***.

Table 2 shows that there were 23 (24.7%) survey participants that indicated this pattern. We observed a fairly uniform distribution of this pattern across small and medium-sized organizations.

External Team: An external consulting organization helps both software provider and customer organizations by creating a customized CDP and then team members in the organization are trained to use and maintain that pipeline. Our results show that a few number of the interviewees' and survey participants' organizations sought external organizations for this purpose.

Team-driven model: In this model, each team in an organization builds and develops their own pipeline to adhere to the needs of the team and project. This model was mainly found by the survey results. When we asked the participants about the formation of a CDP team, the "Other" field was also considered to gather more patterns. 22 survey respondents indicated that their organizations followed this model (i.e., individual team in Table 2). As stated by **R89**, "*Each team has organized their continuous delivery pipeline*", and **R47**, "*Various pipelines are built by engineers and used by themselves*". We also found that in some organizations a central team provides consultancy to all project teams to help project teams to build and manage their own CDP ("*Each team builds its own pipeline with help from a central team*" ***R9***).

Table 2: Survey results on CDP team patterns

CDP team patterns	#	%
Centralized Team	37	39.7
Temporary Team	23	24.7
External Team	3	3.2
Individual Team	22	23.6
Not Available	8	8.6

4.3 Collaboration (RQ2)

Based on the analysis of the interviews' data, we found that besides changing the team structures, organizations are increasingly improving collaboration among teams and team members to effectively initiate and adopt CD practices. We asked the survey respondents to rate how they strongly agree or disagree that the collaboration between teams (e.g., developers, quality assurance team, testers, and operations personnel) has increased in their respective organizations since the adoption of CD practices (See statements S1 in Figure 3). The results indicate that 73.1% of the respondents strongly agreed or agreed with this statement; only 6.4% of the respondents indicated disagreement with the statement S1, and none of them disagreed strongly.

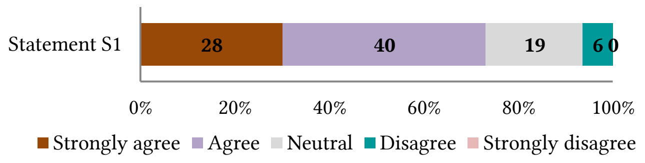

Figure 3: Statement S1- The collaboration between team members has increased in my organization since the adoption of CD.

Through a mandatory open-ended question, we investigated how organizations could foster collaboration among teams. The respondents were expected to specify strategies employed by their organizations for this purpose. The analysis of the provided answers revealed the following practices:

Co-locating teams: The most common strategy to improve collaboration is co-locating teams and discuss, for example, operational issues more often before an application is released to production or customers ("*Dev/Ops/InfoSec team co-location*" ***R17***). The respondents revealed that adopting CD not only needs tighter collaboration between Dev and Ops teams, but also other teams need to be physically close to each other to enable face-to-face communication, faster and easier interaction and knowledge sharing (e.g., "*Placed hardware [team] along with software [team]*" ***R57*** or "*Analysis [team] next to developers*" ***R60***).

Rapid feedback: A few number of the participants emphasized that having shorter feedback loop at each stage in CDP enables teams and team members to partner in producing high quality software. As described by **R19** "*the rapid feedback loop has allowed developers and testers to partner in producing high quality software*". This also allows them to significantly reduce the time between problem identification and problem solving (e.g., "*Shorter loop from feedback to fixes bugs*" ***R38***).

Joint work and shared responsibility: Our results reveal that the speed and frequency demanded by DevOps and CD practices drive the need for a more holistic view, in which team members from each side of the fence are needed to jointly work together and adopt shared responsibility as much as possible. As this quote from **R28** shows: "*Developers are working with operations to make sure their concerns are addressed as part of CD pipeline (i.e., monitoring and health checks)*". The respondent **R33** also confirmed this in the following quote: "*Speaking for a large organization containing dozens of individual cases, it is my experience, however, engineers from each side of the fence need to sit down and discuss what the pipeline actually looks like, what it should look like, and what their respective roles in that pipeline are/should be*". Empowering the culture of shared responsibility is crucial to achieve CD, as shown in this example given by **R88** "*The team as a whole is responsible for the quality of the application, everybody does testing, everybody solves operations problems*". It is argued by the participants that this is a success of a team as a whole, any failure impacts each tier exponentially in terms of cost; hence, the aim is to minimize the failures, particularly before deployment process.

Several survey respondents explained that the overhead of collaboration was sustainably reduced in their organizations by incorporating testing as an integral part of development team instead of testers just being as assessors. For example, **R79** pointed out that "*many testers were trained to be developers and have become valuable members of staff, better developers than those with years of experience*". As indicated by **R68**, a QA team should be paired with the development team to successfully adopt CD: "*CD asks for maximum collaboration between different disciplines of the system, a QA has to pair a lot with developers to understand the delivery to give the signoff*".

Collaboration and communications among team members can considerably increase by establishing cross-functional teams as explained by **R85** "*Cross functional teams became the norm and communication and collaboration increased ten-fold, as teams became more self-organizing*".

Using collaboration tools more often: Several participants indicated that the use of communication and collaboration tools to drive collaborative works between teams has increased since CD adoption (e.g., "*Communication over Slack increased*" ***R84***). Several participants indicated that using common tools and processes across teams in an organization decreases the overhead of collaboration and communication. This enables teams to have cross-collaboration to refine work prior to releasing applications to customers or production environments. A program manager described this vividly: "*Information is being shared in many ways across them [Dev and Ops] and sharing the same Wiki for example in terms of they both get notified when changes are made to documents on the Wiki, using the same JIRA system*" ***P7***.

Increased transparency and awareness: During the interviews, we found that the lack of suitable awareness on status of project (e.g., build status, release status) among team members can be a bottleneck for collaborative work and significantly hinders the CD success. To better understand this challenge, we asked the survey respondents to rate the severity of this challenge through five-point Likert-scale question.

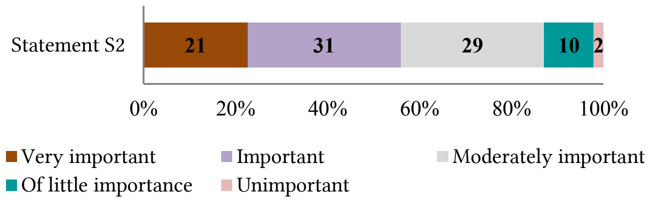

Figure 4: Statement S2- How important is "*lack of suitable awareness on status of project among team members*" in adopting CD.

As shown in Figure 4, 55.9% of the survey participants voted the statement S2 as *very important* or *important.* While only 12 out of 93 participants considered this challenge as *unimportant* or *of little importance.* Besides visibility of build results and test suites execution results, our survey participants emphasized that operations tasks and stuff should be visible and traceable to everyone in team. For example **R67** explained that "*Operations teams have now multi-channel feedback to Dev team (Email, Call, Monitoring Dashboard, Alarms, and Reports)*". **R43** elaborated further this in the following words: "*The operations team has a daily meeting and a Kanban that other teams can go and interact with*".

Empowering and engaging operations personnel: Our survey data shows that the overhead of collaboration and communication between development and operations teams reduced by shifting some of the operations' responsibilities to development team (e.g., "*Development and QA teams are together in a cross-functional team. We also both perform an Op's function, through monitoring and analyzing production behavior*" ***R71***). This situation gives more freedom and time to Ops team to directly and freely collaborate with other team members as stated by **R56** "*Most of the operations work has reduced and they are able to help Dev and QA as they are having [more] time to help*". Giving more power to Ops personnel and engaging them in software development life cycle right from the beginning was referred by a couple of the survey participants as enablers for collaboration. For example, participant **R14** pointed out "*Collaboration between these groups has been high as it's always my intention to involve these groups early in the project lifecycle as possible to ensure the correct parties have their say early in the solution*". A number of the participants mentioned that the interaction between Ops stakeholders and other team members previously used to happen only during production deployment. It has been observed that Ops team became more interactive before each deployment after gaining a voice in development and deployment decisions and ability to influence on design and formation of CDP pipeline. One of the interviewees described this perfectly:

"*I think what we tried to do is to let operations team not only be responsible for operations tasks, and they may also be injecting requirements into build cycle within the project. So they need to be empowered to have equal voice on the team in order to represent their needs and the team is empowered and required to support those needs. This is unlike to work traditional model in the past, where the operations team was a separate team. In our model, operations team was tightly integrated into the development, decisions and planning on the daily basis*" ***P6***.

4.4 Responsibilities (RQ3)

We observed that adopting CD practices changes the responsibilities of some team members. Rather sometime there are more responsibilities that require the acquisition of new skills to align themselves with the spirit of CD practices. For example, **P9** highlighted this change as the follows:

"*So the real responsibility of deployment moved from these different departments to the development team. So development teams become more and more a DevOps team. That's what we tried to do this step by step. So for example, we also tried to integrate database persons in the team; all the database changes are now performed by Dev team*" ***P9***.

This means by adopting CD every function of an organization might be touched, not just development. We were interested in understanding the changes brought about by the adoption of CD in daily work routine of team members. According to data from statement S3 in Figure 5, 56.9% of the survey participants indicated that their responsibilities have changed *somewhat*, *much*, or *very much*. However, of the 23 (24.7%) participants that responded to this statement as *not at all*, more than 60% introduced themselves as consultant or mentioned that their responsibilities have not changed because when they joined their current organizations, CD had already been implemented (e.g., "*No [change], when I joined CD was already adopted*" ***R89***).

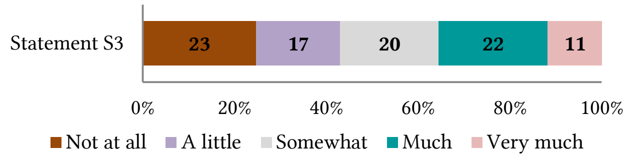

Figure 5: Statement S3- My responsibility has changed after our organization adopted CD practices.

Through a follow-up question, we asked them to explain how their responsibilities have changed (e.g., what new skills they require for practicing CD).

Expand skill-set: Interestingly most of the respondents indicated that they have to constantly learn best practices and new tools for reliable release (e.g., *"[working in CD context] requires familiarity with cloud deployment tools"* ***R24*** or "*focus on tools of CI and CD*" ***R67***). In our survey, we perceived that development team needs to significantly develop their operational skills as well. As the participant **R76** stated that "*Coming from a development side, I had to develop some "ops" skills. When your commit goes automatically to production you've to care about security, on-call and performance of your application*". One of the operational skills that mostly mentioned by the respondents was monitoring and logging skills. Working in CD context necessitates developing monitoring skills and spending more time on monitoring to triage and quickly respond to production incidents. As stated by **R20** "*Ensuring the product stays deployment ready all the time. Each check-in and change gets monitored*" and **R23** "*I have to be more watchful on the deliverables, more stress is on test automation*". Scripting and automation skills were another skills that were referred by several survey participants (e.g., "*Scripting, deploying, automate everything instead of programming only*" ***R58***). We found that CD seeks new bureaucracies to access and manage production environments (e.g., "*Infrastructure and Platform now treated as code [in CD context] and environments defined at last minute*" ***R45***). This helped them to reduce security problems, avoid down time in production environment and better follow ITIL (i.e., Information Technology Infrastructure Library) in transition towards CD practices. In addition, for some of the respondents adopting CD means to understand the whole stack of the application: database, backend, front-end, OS, and build. One of them stated this in these words: "*Skillset required has expanded to more of complete DevOps workflow*" ***R65***. This helped them to further and better be involved in bug fixing (e.g., "*More in depth knowledge of the entire stack - to debug when something fails*" ***R38***).

Adopt new solutions aligned with CD: The findings from the interviews suggest that CD expands and changes the role of architect (e.g., "*You know in terms of overall architecture [for CD], it is not just to know about architecting actual product, it is about architecting the whole picture*" ***P7***) [14]. Our survey results were aligned with this finding; the role and responsibilities of software architects have significantly changed in CD context as only 6 out of 39 architects shared that their responsibility did not change at all. Architects are expected to define and design modern architectures that work with CD process of their organizations (i.e., CD-driven architectures). As explained by **R68** "*As an architect I had to rethink on how we design the systems for continuous delivery*". Another participant validated this change through this quote: "*I have taken on completely new roles; leading architecture work to define internal services to enable these practices [CD practices]*" ***R33***. Understanding and applying microservices architectural style and designing for different deployment models (e.g., Blue-Green deployment) were two main skills and changes reported by the architects to better support CD.

Prioritize tasks: CD greatly helps some team members to concentrate on more valuable tasks (e.g., "*[CD] allows me to focus more on solving business problems instead of release coordination and ceremony*" ***R6***). We also found that building high-quality applications to be deployed frequently and reliably may force team members to spend more time for standardizing their solutions and also improving confidence in the code. This is mainly achieved by performing excessive (automated) testing or shifting part of testing responsibilities to Dev team. One participant stated that "*My responsibilities have shifted from always being able to reproduce every version of code to a model where you always move forward, so I have to think about ways to give trust and confidence in code*" ***R11***. Another example came from participant **R32**, a software engineer, who explained that online functional testing and deployment model checking were two of his new responsibilities towards CD. Furthermore, the focus has shifted more toward automating tasks as much as possible, for example, more concentrating on test automation, creating automated test-cases, and less on tracking down build failures in order to better allocate resources. As explicitly explained by participant **R48** "*My responsibilities are not to do "operations" any more but to think how we are organized and find solutions to provide automation, security and quality for repeatable and trustable deployments*".

5. THREATS TO VALIDITY

One of the possible threats to validity of empirical findings is related to the appropriateness of data collection and sampling approaches. For the objectives of our study, it was important to attract the participants with the relevant background and experience (e.g., DevOps and CD). We addressed this validity threat by applying strict criteria for the potential participants of this study. We recruited the participants purposefully (e.g., seeking for potential practitioners and rigorously reviewing their public profiles) for both parts of this study. We added our participation criteria at the beginning of the survey so only the relevant practitioners could participate. We are confident that most of the participants had the right expertise for taking part in our study. We ensured that the participants of this study held different roles (e.g., software architect, DevOps engineer, and developer) to avoid bias in results by a specific role. In order to alleviate memory bias [34] (i.e., not remembering all the details by the participant), we asked the participants to share their inputs about the most recent projects or clients. The interview questions were shared with the interviewees beforehand in order to enable them to reflect and articulate the related stories.

Another limitation of our study could be the suitability and validity of the interviews' and survey's questions. Whilst the first author designed all the questions, they were rigorously reviewed and verified by the other authors and few practitioners. Some of the questions also got improved based on the feedback we collected from the participants during the study. Conducting semi-structured interviews enabled us to get engaged in discussions with the interviewees and collect rich and elaborate responses. The emerged findings from the interviews led us to design survey questions. Though most of the survey questions were close-ended, we provided an opportunity to the participants to briefly share any other ideas they might have had about the topics of the study.

Utilizing two different data sources has enabled us to triangulate the findings and reduce researchers' bias. The survey questions were derived from the interviews' findings that were reported by more than one participants (e.g., at least 2). This strategy helped us to alleviate negative impact of any possible subjective viewpoint of the results. The qualitative findings are highly dependent upon researchers' interpretations and could be influenced by personal understanding and opinions. Whilst the first author has mainly performed the analysis of the gathered data, the other authors verified the coded data during frequently discussion sessions organized to discuss and clarify any doubts.

Finally, a potential threat to our study could be generalizability of the findings. We assert that our sampling strategy and employing mixed-method research approach have highly improved generalizability of our results. Our interviewees were purposefully invited with different backgrounds (e.g., roles and experiences) from variety of organizations (i.e., variant in terms of size and domain) located in different geographies. This sampling technique helped us to capture more general context. We have augmented and generalized the interviews' findings through surveying a larger pool of practitioners.

6. DISCUSSION AND CONCLUSIONS

DevOps paradigm stresses higher coordination and collaboration between members involved in software delivery to release high quality software faster and more reliably [21]. It is asserted that collaboration is one of the key dimensions of DevOps [22; 31] for supporting the changes in organization' structure and culture as a result of adopting DevOps. In many organizations, Development (Dev) and Operations (Ops) teams form silos that are possibly located in separate departments [31]. Whilst this structure was motivated by traditional methodologies (e.g., waterfall), it is not suitable for recent software development practices that simultaneously deal with agility, and maintaining software on different environments [22]. Iden et al. [23] highlight that effective cooperation between Dev and Ops teams has great impact on the quality of the final product. Any shortcomings in interaction of these members usually manifest in problems such as excluding IT operations from requirement specifications, poor communication and information flow, and lack of knowledge transfer [23].

Lwakatare et al. [29] indicate that collaboration can be enforced through practices such as broadening skill-set, information sharing and shifting responsibilities among these members. Nevertheless, implementing these practices demand changes in team structures, required responsibilities, the work culture of organization and mindset of team members [29; 31]. Some researchers have identified best practices to implement these changes (e.g., team structure). For example, Humble and Molskey [22] suggest re-architecting software product in form of strategic services, and assigning each service to a small cross-functional team who takes the full ownership of it during whole development lifecycle. Our research has been motivated by the need of empirically studying and understanding organizational practices promoting collaboration principle of DevOps. Our empirical study's findings have identified four key patterns for structuring Dev and Ops teams to effectively initiate CD practices. The popularity and applicability of these patterns vary given the organizational context (e.g., hierarchies and size). There is a higher tendency among large organizations to initiate CD practices while maintaining separate Dev and Ops units. These organizations have promoted collaboration among their Dev and Ops teams through different means (e.g., collocating members and employing facilitator team); yet they have not drastically changed organizational structure for breaking the silos. We have observed that small organizations have more flexibility and tendency to employ different patterns aimed at merging these units in form of unified multidisciplinary team(s). We have pointed out that this difference may be rooted in the challenges that organizations face in adopting CD ideally with unified multidisciplinary teams. We can enumerate some of these challenges: availability of highly skilled members to form multidisciplinary team, possibility of re-structuring units/ departments, (re) architect software product to independent units (e.g., services) for assigning to multidisciplinary teams, and having highly cooperative organizational culture for forming and running unified teams. We speculate that larger organizations may face more restrictions to change established practices and address these challenges. Future research can extend our findings and investigate the role of different contextual factors (e.g., global distribution of sites, business domain, and type of products) in adopting different team structures when moving to CD practices.

Our study has revealed several organizational practices improving collaboration among teams and team members to effectively implement CD. Our findings in this regard are aligned with the previous research [21; 22; 31] while providing significantly additional insights. It is evident that sharing responsibilities of software delivery with *all* members [22] could promote coordination and collaboration in a team. We observed the practice of rotating roles [31] between developers and operations staff, i.e., involving developers in testing and QAs in development tasks. Our findings have highlighted the significant role of visibility and awareness of a project status for improving collaboration in a team and successfully adopting CD. Humble and Farley [21] recommend using big, ubiquitous dashboards in each team room to visualize status of builds and sharing feedback with everybody. Our participants also indicated raising awareness in teams by involving operations staff in daily meetings [31] and interacting with Kanban board. We have discussed that collaboration among team members not only can be improved through processes, but also by provisioning appropriate tool support. Some studies (e.g., [32]) have demonstrated that while organizations extensively utilize tool chains for building deployment pipeline, there is less focus on technologies facilitating communication and knowledge sharing in teams. Future research should explore the possibilities for promoting communication and collaboration through tools.

Implementing CD practices demands skill-set and knowledge that are either brand new (e.g., tools for automating CD process), or lie at the intersection of development and operations responsibilities. Similar to [31], our study reveals that adopting CD broadens the scope of responsibilities and skill-set. It is evident that these changes are particularly significant for developers who sometimes take larger shares in operations activities [31]. Whilst developers are expected to take active part in deployment for successful adoption of CD, it should not demolish operations' functions. Broadening responsibilities of developers to a larger extent may negatively impact their productivity in core tasks. Shifting extensive amount of operations' responsibilities to developers could cause fear of losing jobs for Ops team and may negatively affect success of transition to CD. Organizations should extensively promote knowledge sharing among team members to complement areas of skill-set and collaboratively work towards a shared goal.

7. ACKNOWLEDGMENTS

The authors would like to thank all participants. This work is partially supported by Data61, a business unit of CSIRO, Australia. The first author is also supported by Australian Government Research Training Program Scholarship.

8. REFERENCES

[1] There's No Such Thing as a "Devops Team", https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/.

[2] What Team Structure is Right for DevOps to Flourish? http://web.devopstopologies.com/.

[3] What's the Best Team Structure for DevOps Success? https://puppet.com/blog/what%E2%80%99s-best-team-structure-for-devops-success.

[4] 2015. *2015 State of DevOps Report. Available at: https://puppetlabs.com/2015-devops-report [Last accessed: 5 October 2015].*

[5] BASS, L., WEBER, I., and ZHU, L., 2015. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.

[6] BITTNER, K. and BUNTEL, T., Overcoming Organizational Obstacles to DevOps and Continuous Delivery, https://xebialabs.com/community/webinars/overcoming-organizational-obstacles-to-devops-and-continuous-delivery/.

[7] CALLANAN, M. and SPILLANE, A., 2016. DevOps: Making It Easy to Do the Right Thing. *IEEE Software 33*, 3, 53-59.

[8] CLAPS, G.G., BERNTSSON SVENSSON, R., and AURUM, A., 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology 57*, 21-31.

[9] CRESWELL, J.W. and CLARK, V.L.P., 2007. *Designing and conducting mixed methods research*. SAGE Publications.

[10] CRUZES, D.S. and DYBA, T., 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)* (22-23 Sept. 2011 2011), 275-284.

[11] DINGSØYR, T. and LASSENIUS, C., 2016. Emerging themes in agile software development: Introduction to the special section on continuous value delivery. *Information and Software Technology 77*, 56-60.

[12] EASTERBROOK, S., SINGER, J., STOREY, M.-A., and DAMIAN, D., 2008. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering* Springer, 285-311.

[13] FITZGERALD, B. and STOL, K.-J., 2017. Continuous Software Engineering: A Roadmap and Agenda. *Journal of Systems and Software 123*.

[14] FORD, N., Continuous Delivery for Architects, Available at: http://nealford.com/downloads/Continuous_Delivery_for_Architects_Neal_Ford.pdf [Last accessed: 20 October 2016].

[15] FOWLER, M., 2013. Continuous Delivery. Available at: http://martinfowler.com/bliki/ContinuousDelivery.html [Last accessed: 21 October 2015].

[16] FRANÇA, B.B.N.D., JUNIOR, H.J., and TRAVASSOS, G.H., 2016. Characterizing DevOps by Hearing Multiple Voices. In *Proceedings of the 30th Brazilian Symposium on Software Engineering* (Maringa-PR, Brazil2016), ACM, 53-62.

[17] GOODMAN, L.A., 1961. Snowball Sampling. *Annals of Mathematical Statistics 32*, 1, 148-170.

[18] GOUSIOS, G., STOREY, M.-A., and BACCHELLI, A., 2016. Work practices and challenges in pull-based development: the contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering* (Austin, Texas2016), ACM, 285-296.

[19] HOVE, S.E. and ANDA, B., 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, 23-32.

[20] HUMBLE, J., Continuous Delivery vs Continuous Deployment, Available at: https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/ [Last accessed: 1 March 2016].

[21] HUMBLE, J. and FARLEY, D., 2010. *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

[22] HUMBLE, J. and MOLESKY, J., 2011. Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal 24*, 8, 6.

[23] IDEN, J., TESSEM, B., and PÄIVÄRINTA, T., 2011. Problems in the interplay of development and IT operations in system development projects: A Delphi study of Norwegian IT experts. *Information and Software Technology 53*, 4, 394-406.

[24] KITCHENHAM, B., PICKARD, L., and PFLEEGER, S.L., 1995. Case studies for method and tool evaluation. *IEEE Software 12*, 4, 52-62.

[25] KITCHENHAM, B.A. and PFLEEGER, S.L., 2008. Personal Opinion Surveys. In *Guide to Advanced Empirical Software Engineering*, F. SHULL, J. SINGER and D.I.K. SJØBERG Eds. Springer London, London, 63-92.

[26] KRUSCHE, S. and ALPEROWITZ, L., 2014. Introduction of continuous delivery in multi-customer project courses. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India2014), ACM, 335-343.

[27] LEPPANEN, M., MAKINEN, S., PAGELS, M., ELORANTA, V.-P., ITKONEN, J., MANTYLA, M.V., and MANNISTO, T., 2015. The Highways and Country Roads to Continuous Deployment. *IEEE Software 32*, 2, 64-72.

[28] LUKE, E. and PRINCE, S., 2016. No One Agrees How to Define CI or CD. Available at: https://blog.snap-ci.com/blog/2016/07/26/continuous-delivery-integration-devops-research/, [Last accessed: 1 August 2016].

[29] LWAKATARE, L., KUVAJA, P., and OIVO, M., 2015. Dimensions of DevOps. In *Agile Processes, in Software Engineering, and Extreme Programming*, C. LASSENIUS, T. DINGSØYR and M. PAASIVAARA Eds. Springer International Publishing, 212-217. DOI= http://dx.doi.org/10.1007/978-3-319-18612-2_19.

[30] LWAKATARE, L.E., KARVONEN, T., SAUVOLA, T., KUVAJA, P., OLSSON, H.H., BOSCH, J., and OIVO, M., 2016. Towards DevOps in the Embedded Systems Domain: Why is It So Hard? In *49th Hawaii International Conference on System Sciences (HICSS)*, 5437-5446.

[31] LWAKATARE, L.E., KUVAJA, P., and OIVO, M., 2016. An Exploratory Study of DevOps Extending the Dimensions of DevOps with Practices. In *Proceedings of the The Eleventh International Conference on Software Engineering Advances (ICSEA)* (2016), IARIA, 91-99.

[32] MÄKINEN, S., LEPPÄNEN, M., KILAMO, T., MATTILA, A.-L., LAUKKANEN, E., PAGELS, M., and MÄNNISTÖ, T., 2016. Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises. *Information and Software Technology 80*, 175-194.

[33] MOONEY, M., Continuous Deployment For Practical People, https://www.airpair.com/continuous-deployment/posts/continuous-deployment-for-practical-people.

[34] MURPHY-HILL, E., ZIMMERMANN, T., BIRD, C., and NAGAPPAN, N., 2015. The Design Space of Bug Fixes and How Developers Navigate It. *IEEE Transactions on Software Engineering 41*, 1, 65-81.

[35] NEELY, S. and STOLT, S., 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *Proceedings of the Agile Conference (AGILE)* (5-9 Aug 2013), 121-128.

[36] NYBOM, K., SMEDS, J., and PORRES, I., 2016. On the Impact of Mixing Responsibilities Between Devs and Ops. In *Agile Processes, in Software Engineering, and Extreme Programming: XP 2016, Edinburgh, UK, May 24-27, 2016.*, H. SHARP and T. HALL Eds. Springer International Publishing, 131-143.

[37] PALINKAS, L.A., HORWITZ, S.M., GREEN, C.A., WISDOM, J.P., DUAN, N., and HOAGWOOD, K., 2015. Purposeful Sampling for Qualitative Data Collection and Analysis in Mixed Method Implementation Research. *Administration and Policy in Mental Health and Mental Health Services Research 42*, 5, 533-544.

[38] PHILLIPS, A., SENS, M., DE JONGE, A., and VAN HOLSTEIJN, M., 2015. *The IT Manager's Guide to Continuous Delivery: Delivering business value in hours, not months*. XebiaLabs.

[39] REED, J.P., The business case for continuous delivery, Available at https://www.atlassian.com/continuous-delivery/business-case-for-continuous-delivery, [Last accessed: 12 July 2016].

[40] SAVOR, T., DOUGLAS, M., GENTILI, M., WILLIAMS, L., BECK, K., and STUMM, M., 2016. Continuous deployment at Facebook and OANDA. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas2016), ACM, 21-30.

[41] SHAHIN, M., ALI BABAR, M., and ZHU, L., 2016. The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Ciudad Real, Spain2016), ACM.

[42] STÅHL, D., HALLÉN, K., and BOSCH, J., 2016. Continuous Integration and Delivery Traceability in Industry: Needs and Practices. In *Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Aug. 31 2016-Sept. 2 2016), 68-72.

[43] THIELE, A., 2014. Continuous Delivery: An Easy Must-Have for Agile Development, Available at: https://blog.inf.ed.ac.uk/sapm/2014/02/04/continuous-delivery-an-easy-must-have-for-agile-development/ [Last accessed: 10 July 2016].

[44] WEBER, I., NEPAL, S., and ZHU, L., 2016. Developing Dependable and Secure Cloud Applications. *IEEE Internet Computing 20*, 3, 74-79.

[45] WETTINGER, J., BREITENBÜCHER, U., FALKENTHAL, M., and LEYMANN, F., 2016. Collaborative gathering and continuous delivery of DevOps solutions through repositories. *Computer Science - Research and Development*, 1-10.

[46] WETTINGER, J., BREITENBÜCHER, U., KOPP, O., and LEYMANN, F., 2016. Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel. *Future Generation Computer Systems 56*, 317-332.