In Proceedings of 11th international Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, Canada, 2017

Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges

Mojtaba Shahin [a], Muhammad Ali Babar [a], Mansooreh Zahedi [b], 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, Australia
mojtaba.shahin@adelaide.edu.au, ali.babar@adelaide.edu.au, mzah@itu.dk, liming.zhu@data61.csiro.au

***Abstract— Context:* A growing number of software organizations have been adopting Continuous DElivery (CDE) and Continuous Deployment (CD) practices. Researchers have started investing significant efforts in studying different aspects of CDE and CD. Many studies refer CDE (i.e., where an application is *potentially* capable of being deployed) and CD (i.e., where an application is *automatically* deployed on *every update*) as synonyms and do not distinguish them from each other. Despite CDE being successfully adopted by a large number of organizations, it is not empirically known why organizations still are unable or demotivated to have *automatic* and *continuous* deployment (i.e., CD practice). *Goal*: This study aims at empirically investigating and classifying the factors that may impact on adopting and implementing CD practice. *Method*: We conducted a mixed-method empirical study consisting of interviewing 21 software practitioners, followed by a survey with 98 respondents. *Results*: Our study reveals 11 confounding factors that limit or demotivate software organizations to push changes *automatically* and *continuously* to production. The most important ones are "lack of automated (user) acceptance test", "manual quality check", "deployment as business decision", "insufficient level of automated test coverage", and "highly bureaucratic deployment process". *Conclusion*: Our findings highlight several areas for future research and provide suggestions for practitioners to streamline deployment process.**

Keywords— continuous delivery; continuous deployment; empirical study

I. Introduction

"*We do continuous delivery even to on-premise environments. Using continuous delivery, we would not be pushing out every day. We might only push out new release to client site every three months. We are still using continuous delivery practice to keep software deployable and releasable. I think it is quite important to distinguish continuous delivery and continuous deployment*" ***Consultant (P18)***.

"*You can apply continuous delivery and not implement continuous deployment yet (e.g., because customer has security constrains to deploy remotely)*" ***Team Lead (R97)***.

Development and Operations (DevOps) is an approach to solve a disconnect between development and operations teams by promoting collaboration, communication, and integration between them [1]. The increasing need for improvement in the business competitiveness and performance has compelled many IT organizations to adopt DevOps [1, 2]. Continuous DElivery (CDE) and Continuous Deployment (CD) are key DevOps practices to suitably achieve this goal by enabling IT organizations to accelerate delivering high-quality value to customers [1]. It should be noted that there is no consensus on the definitions of CDE and CD practices in both academic and industry communities as software organizations may interpret and employ them interchangeably [3-6]. The main goal of CDE practice is to keep an application always at deployable state [7]. CD practice is an extension to CDE, which *automatically* and *continuously* releases the application (changes) to production environment after successfully passing all automated tests and quality checks. In contrast to CDE (i.e., a pull-based approach), CD practice (i.e., a push-based approach) does not have any manual steps or decision making process for *when* and *what* to release: once developers commit a change, the change is immediately pushed to production without human intervention [8, 9]. It is argued that CD practice enables organizations to immediately gain feedback from users and production use [8, 10].

Whilst several studies have investigated the challenges and issues for adopting and implementing CDE [11-13] and CD practices [14-17], they usually use CDE and CD as synonyms [18, 19]. However, there is the other side of the coin. A recently published SLR on CDE [18] reveals that existing literature uses the term CD while they actually refer to CDE, because they do not include or provide fully automatic deployment to production. Furthermore, the SLR could not find highly relevant scientific literature on CD implementation. Recently industrial communities [20-22] have underlined the challenges, experiences, and lessons learnt in moving from CDE (i.e., where an application is *potentially* capable of being deployed) to CD (i.e., where the application is *automatically* deployed to production on *every update*). In addition, two studies [23, 24] reveal that delivery (i.e., CDE) and deployment (i.e., CD) capabilities of software organizations may be different as for example there might be a tension between software quality and deployment frequency. In [23], a new line of research is explicitly sought to explore the reasons for this difference. To the best of our knowledge, none of the previously published literature reports this issue and specifically distinguishes between the challenges of adopting CDE and CD. This paper aims at empirically investigating and classifying confounding factors that particularly impact on adopting and implementing CD practice: despite software *potentially* is production-ready (i.e., CDE practice), there are still factors that limit or demotivate organizations to *continuously* and *automatically* ship code changes from development into production without human intervention (i.e., CD practice). To help closing this literature gap, our paper investigates the following research question:

***RQ**: What factors do limit or demotivate software organizations to move from continuous delivery to continuous deployment?*

To answer this research question, we conducted a mixed-method empirical study that collected data from in-depth interviews with 21 experts from 19 organizations and a survey of 98 software practitioners. Our analysis reveals 11 factors that are confounders to truly adopt CD practice, including: "lack of fully automated (user) acceptance test", "manual quality check", "deployment as business decision", "insufficient level of automated test coverage", "highly bureaucratic deployment process", "lack of efficient rollback mechanism", "dependency at application level", "demotivated customer", "customer environment", "domain constraints", and "manual interpretation of test results". Our findings have significant differences to the existing studies [11-17]: (1) our findings come from interviewing 21 experts in CDE and CD and a survey of 98 software professionals from a wide range of organizations in terms of size, domain, and the way of working rather than one practitioner's own observations [11, 12] or a single case company [13, 16] and a particular domain [14]. The study [14] focuses only on adopting DevOps in embedded systems and the studies [11, 12] only identify the challenges and issues of adopting CDE and CD based on the experience of authors. (2) This paper discusses the current state of automation support in software industry to adopt CD, which has not been reported in the previous work. (3) Our study reports the first large-scale empirical investigation of the challenges of having *automatic* and *continuous* deployment (i.e., CD).

The rest of this paper is organized as follows: Section II describes the research method. Section III reports our findings. We present potential threats of this study in Section IV. Section V reflects a discussion on findings. In Section VI, we summarize related work. Finally, the paper is concluded in Section VII.

II. Research Method

Considering the exploratory nature of our research question, we chose to use a mixed-method research approach with *sequential exploratory strategy* [25]. We first collected qualitative data from 21 in-depth, semi-structured interviews and then we assessed and quantified the findings through surveying 98 software professionals [25, 26]. This section describes the research protocol that was strictly followed for this study.

A. Interviews

Protocol: For this study, we utilized the data gathered through interviews with 21 practitioners from 19 organizations based in 9 countries. The interviews were semi-structured with almost 40 open-ended questions. As all of the interviewees were located in different parts of the world, it was not possible for us to do face-to-face interview. Hence, all interviews except two (i.e., email-based interview) were conducted via Skype. To engage the interviewees in an in-depth discussion and thoroughly gather their perspectives, we sent the interview questions to them beforehand [27]. During the interviews, we made some adjustments to some questions based on the responses and comments of the interviewees. All of the interviews were conducted by the first author and were audio-recorded and transcribed for an in-depth analysis. The relevant parts of the interviews for this study are described as follows: first part briefed the high level objectives of the study to the interviewees. In the second part, we precisely defined CDE and CD terms for the interviewees and explained what differences exist between them. Next, questions related to participant's background were asked (e.g., *what is your role and responsibilities in the project team?*). Forth, the participants explained challenges, their personal experiences and concerns around moving from CDE to CD, and why they were still unable or demotivated to have fully *automatic* deployment to production. In the last part, questions related to deployment pipeline were asked (e.g., *is your deployment pipeline fully automated or not? why?*). We finished the interviews by asking the interviewees "*Is there any comment or issue you want us to know?*"

Participants: We used purposive sampling strategy [28] to recruit representatives from the organizations that adopted or were adopting CDE or CD practices or experts who were part of organizations doing consulting in these areas. We followed three steps to identify the participants: (i) we identified the potential participants through personal network, and by exploring the list of speakers and attendees of industry-driven conferences on DevOps/CD (e.g., DevOps Enterprise Summit[1]). (ii) We strictly analyzed their profiles to understand whether they had the right kind of experiences and expertise to participate in our study. (iii) Then, we directly sent an invitation email to them. To maximize participation, the interviewees were promised to receive a free copy of a book on DevOps: "*DevOps: A Software Architect's Perspective*" [1]. We purposively targeted software professionals with different levels of seniority, different types of experiences and holding different roles, from diverse organizations in terms of domains and sizes. "*Snowballing technique*" has been used to expand the number of participants, in which at the end of each interview we asked the interviewee to introduce potential interviewees [29].

Data analysis: We employed conceptualized thematic analysis method in software engineering to analyze interview transcripts [30]. We supported the qualitative analysis process by NVivo[2] software (i.e., a qualitative data analysis tool). This allowed a systematic and more convenient analysis and comparison of emerging themes. Whilst first author carried out data analysis, the third author assessed all emergent themes to confirm them and identify any other potential themes. Following five steps of the conceptualized thematic analysis method, first we read the interview transcripts line-by-line to extract key points of each interview and transferred them to NVivo software. Second, based on the detailed answers to the interviews questions, we created *initial codes* for later analysis. Third, the codes identified in last step were clustered into *potential themes*. Next step involved re-evaluating the extracted themes against each other to merge presumably related themes or exclude the themes with low evidence support. At the end of this step, we generated a *higher-order model of themes*. Last step consisted of assessing trustworthiness of each core theme and assigning a name to each.

Interviewees Characteristics: In total, 21 practitioners (i.e., indicated by **P1** to **P21**) participated in the interviews. All participants were male. Regarding the interviewees' experiences in software development, over 65% have more than 10 years, five have 6-10 years and two have 1-5 years of experience. 7 out of 21 participants were currently in the role of architect, followed by consultants (4 out 21, %19). The rest of them were executives

[1] *events.itrevolution.com*

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

(e.g., CTO, 2), team leads (2), program managers (2), developer (1), DevOps engineer (1), operations engineer (1), and software engineer (1). The interviewees' organizations were from different software domains including consulting and IT services (8), financial (2), telecommunication (2), games (2). 9 interviewees worked in large organizations (>1000 staff), 7 in medium-sized (100-1000 staff) and 5 in small ones (<100 staff).

B. Survey

Protocol: We designed an online survey including qualitative and quantitative questions following the guidelines developed by Kitchenham and Pfleeger [31]. Based on the themes emerged from the interviews, we formulated survey questions. The survey allowed us to assess, complement and generalize the interviews' findings with larger number of respondents. Similar to the interviews, the survey began with definition of research objectives and explaining eligibility criteria for participation. We precisely defined CDE and CD at beginning of the survey instrument. The relevant survey questions used for this study were composed of 4 demographic, 3 five-point Likert-scale, 2 single-choice, 2 multiple-choice and 4 open-ended questions. For multiple- and single-choice questions, "Other" field was added to unveil additional perspectives and thoughts [32]. Answering all questions was mandatory. Likert-scale questions aimed at collecting the participants' views on three types of statements: (1) how they agreed or disagreed with a statement (i.e., from *strongly agree* to *strongly disagree*); (2) how important the challenge reported in a statement was (i.e., from *very important* to *unimportant*); (3) how they scored a statement (i.e., from *1* to *5*).

Participants: We employed three recruitment methods for our survey. Initially, we publically advertised the survey to social media, for example DevOps/CD related groups on LinkedIn. Secondly, an invitation letter was sent to about 4000 GitHub users via email and invited them to complete the survey. In the email invitation and survey preamble, the participants were asked to forward the survey to their colleagues. Despite motivating practitioners by raffling for five DevOps books (i.e., "*DevOps: A Software Architect's Perspective*"), we were not successful in recruiting practitioners using the first two methods. About 15% of all responses came from applying the aforementioned methods. Previous research has also indicated that these methods may fail to incentivize large number of populations [33]. Hence, we decided to reach out highly relevant practitioners by applying the interviewees' recruitment process: identifying highly relevant practitioners (e.g., speakers and attendees of industry-driven conferences on CD), carefully examining their backgrounds and expertise, and sending an invitation email to them. We directly contacted 487 software practitioners through email. Our sample size was not similar to [32, 33], so it was not feasible to calculate a response rate for our survey. We eventually received 98 survey responses from the invited participants.

Data Analysis: The data collected from the Likert-scale and close-ended questions was analyzed using descriptive statistics [33]. We applied conceptualized thematic method (as discussed in Section II.A) to analyze the responses to open-ended questions: first author conducted qualitative analysis and then the third author assessed all the emergent themes.

Survey Participants Characteristics: We received responses from 98 participants (i.e. indicated by **R1** to **R98**). Majority of the survey participants were architects (40), followed by DevOps engineers (12), consultants (10), and team leads (8). The rest were developers (7), software engineers (6), executives (e.g., director, 3), operations engineers (3), etc. 75.5% of the survey participants had more than 10 years of experience in software industry, 14.3% 6-10 years, 7.1% 3-5 years, and 3.1% 1-2 years. 39 practitioners from large, 31 from medium-sized and 28 from small organizations completed the survey. Similar to the interviewees, the survey participants came from very diverse organizations in terms of domain including consulting and IT services (36), financial (10), e-commerce (10), and telecommunication (6).

III. Findings

First, we present our findings regarding differences in practicing CDE and CD in software industry. Next, as automation is a key component of CD practice, we describe the current state of automation support in software indusry in this regard. Then, we report confounding factors in moving from CDE to CD.

A. Practicing CDE vs. CD in Industry

We aimed at understanding the differences in practicing CDE and CD in software industry and also assessing the maturity of CDE and CD practices in our participants' organizations. To this end [2, 15, 23], both the interviewees and the survey participants were asked two questions: (1) *on average, how often your applications are in releasable state or production-ready?* This question, to a large extent, indicates how successfully an organization adopts and implements CDE practice. To measure CD success in an organization, we asked (2) *on average, how often do you deploy your applications to production environment?* Figure 1 shows that almost 53.7% (64 out of 119 (21+98)) of the participants indicated that on average the applications in their respective or client organizations were in deployable-state *multiple times a day* or *daily*, indicating they were successful to truly implement CDE. However, CD success was lower as in total 43 out of 119 (36.1%) participants indicated that the application changes were automatically pushed *multiple times a day* or *daily* to production. This finding can have twofold implications: First, it shows that compared to the organizations studied in [15, 23], our participants' organizations were more successful in implementing CDE and CD practices. That means our findings came from reliable sources. Second, it reveals that practicing CDE and CD is quite different in software industry: despite CDE being successfully adopted by organizations, there might be factors that limit or demotivate them to have *automatic* and *continuous* deployment to production (i.e., CD practice). Section IV.C reports theses confounding factors.

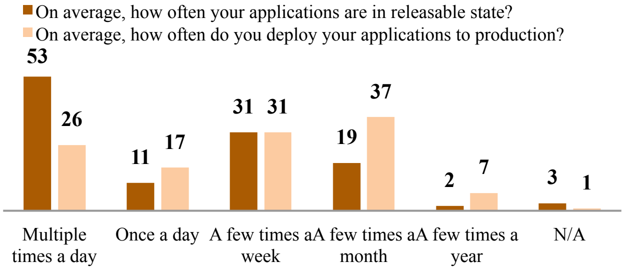

Fig. 1: How continuous delivery and deployment are implemented – aggregated results of interviews and survey

B. Current State of Automation Support in Continuous Deployment Pipeline

Continuous Deployment Pipeline (CDP) (aka. continuous delivery pipeline) plays a significant role in helping organizations to achieve *continuous* and *automatic* deployment [34]. This means that the success of practicing CD in an organization heavily relies on degree of automation support in CDP [35]. A fully automated CDP enables organizations to automatically build, test, configure and deploy new features to production. Therefore, we were interested in understanding how automation is supported in CDPs in software industry. The survey respondents were asked to score their CDPs in terms of automation on a 1-5 scale (i.e., from *1-completely manual* to *2-completely automated*). The data from Figure 2 shows that over 70% (69 out 98) of the survey respondents scored their CDPs *4* or *3*, which can be considered as semi-automated CDPs. Surprisingly, only 19 out of 98 the respondents indicated that they had fully automated CDP to transfer the changes form repository to production.

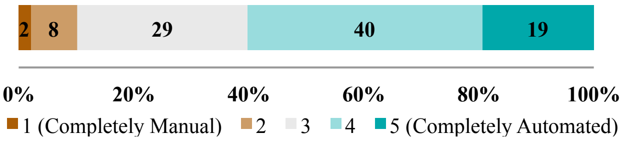

Fig. 2: **Statement 1**: How would you grade your continuous deployment pipeline in terms of automation?

Through a five-point Likert-scale question, we asked the survey participants to indicate how strongly they agree or disagree with this statement (S2): "*we have the right tools to set up fully automated continuous deployment pipeline*". From the Figure 3, we find that whilst 25.5% of the respondents *strongly* agreed, 43.4% agreed that there are right tools for this purpose. Other respondents were divided: 18.3% were neutral, 2% strongly disagreed and 10.2% disagreed with the statement S2.

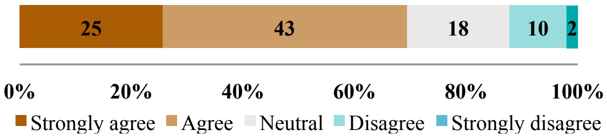

Fig. 3: **Statement 2**: We have the right tools to set up fully automated CDP.

Furthermore, we intended to explore what stages of a CDP may more or less support automation. Typically, a CDP is composed of explicit stages (e.g., build) to push code from source code repository to production [1, 35]. It should be noted that there is no standard or single pipeline as organizations may design and implement their own CDPs with different stages and diverse tools [35]. Through a multiple-choice question, we asked the survey participants which of the following stages constitutes their CDPs: "version control", "build", "continuous integration", "artifact repository management", "unit/integration testing", "acceptance testing", "production deployment", "configuration and provisioning", "log management and monitoring", and "containerization". We also included "Other" field to collect any other stages in a CDP. As Figure 4 shows, "version control", "build", "unit/integration testing", "continuous integration" and "production deployment" were the most common stages of CDPs in software industry. However, "containerization" was the least commonly mentioned stage in CDPs as only 37 survey participants stated that their CDPs include "containerization" stage. It should be noted that not all stages are compulsory in a CDP as only 19 survey participants indicated that the CDP in their respective organizations include all of the abovementioned stages. Figure 4 demonstrates that only 5 survey participants indicated the "Other" field. One respondent pointed out that each application has its own CDP; therefore, there is high variability from application to application. Two others referred to "configuration and provisioning" stage in different ways (e.g., "*Cloud Management and Self-Service*" ***R80*** and "*Automated Provisioning of Environments*" ***R90***). Through two open-ended questions, we asked the survey respondents which of the above-mentioned stage(s) have the most and the least automation support respectively. The responses for these questions indicate that "acceptance testing", "production deployment", and "configuration and provisioning" were the stages that had the least automation support respectively. In contrast, "build", "continuous integration" and "unit/integration testing" stages got the most automation support. Our analysis has revealed that the organizations with fully automated CDPs were much more successful to achieve *frequent* and *automatic* deployment than those with semi-automated CDPs. The responses to these two open-ended questions were fed into our analysis process, where applicable, to explore why some CDP's stages had less automation support and how lack of fully automated CDP limited some participants' organizations to truly adopt CD (See Section III.C).

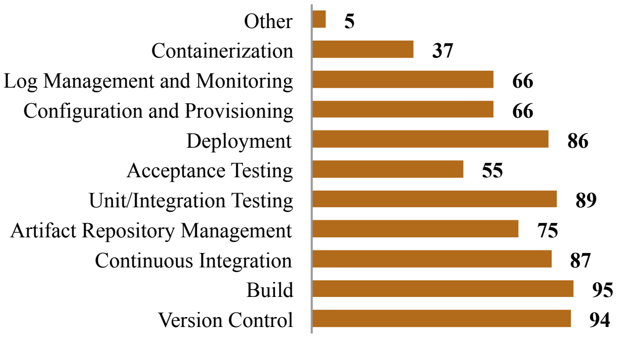

Fig. 4: Stages of Continuous Deployment Pipeline

C. *Moving from CDE to CD*

This section reports the confounding factors in moving from CDE to CD, which are extracted from the interviews and the survey open-ended questions. We also assess and quantify these factors by indicating the number and percentage of the survey respondents who experienced these factors (See Table I).

TABLE I. Summary of confounding factors in moving from CDE to CD

Confounding Factors	**#**	**%**
F1. Lack of fully automated user acceptance test	**43**	**43.9**
F2. Manual quality check	**42**	**42.9**
F3. Deployment as business decision	**41**	**41.8**
F4. Insufficient level of automated test coverage	**34**	**34.7**
F5. Highly bureaucratic deployment process	**31**	**31.6**
F6. Lack of efficient rollback mechanism	**24**	**24.5**
F7. Dependency at application level	**23**	**23.5**
F8. Demotivated customer	**19**	**19.4**
F9. Customer environment	**16**	**16.3**
F10. Domain constraints	**15**	**15.3**
F11. Manual interpretation of test results	**11**	**11.2**

1) Lack of fully (user) automated acceptance test

We found that one of the major changes that usually would happen during transition to CD is to identify reworks and eliminate their root causes effectively. An often-heard reason for reworks in software development process was manual testing. Several interviewees stated that an extensive effort and time in transition to both CDE and CD practices have been spent on automating existing manual tests (e.g., "*From a technical* *perspective let's say since you have to reduce time and move fast, you have to care about testing. The problem is that you can't automate everything because it sounds time consuming*" ***P13***). We were interested in gathering viewpoints of the practitioners in this regard in a quantifiable scale. Hence, we asked the survey participants (n=93) to rank how important was the challenge of "*lack of fully test automation*" in CD adoption, so because of which they faced serious challenges in *automatic* and *continuous* deployment. Figure 5 indicates that 78.7% of the survey's respondents rated the severity of this challenge as *very important* or *important*.

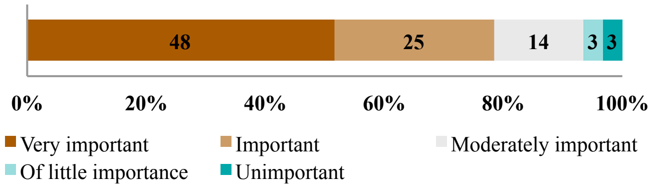

Fig. 5: **Statement 3**: How important is "*lack of fully test automation*" in adopting CD and put you in trouble to have automatic deployment

The interviewees disclosed that they considerably succeeded in automating units and integration tests, but automating the tests occurring at the end of development process such as acceptance test and performance test remained the main challenge and it took heavy workloads and time of their own organizations. As **P9** commented:

"They [organization] really often have challenges to get [tests automated] done, for acceptance tests most of time it is not easy to fully automate" ***P9***.

Our survey's results were aligned with our interviews' findings as "lack of fully user acceptance test automation" has presented the most confounding factor (43 out of 98, 43.9%) for CD success. The survey participants shared the following reasons why (user) acceptance tests were or could not be fully automated.

a) Too much effort with low gain

Surprisingly, many survey participants mentioned concerns about the potential benefits of automating acceptance test at large scale compared to its associated complexities and costs (e.g., "*I'm guessing it [automating user acceptance test] is seen as high effort for low gain*" ***R61***). That is why in some organizations there is not enough demand for this purpose. Furthermore, some believed that acceptance testing should involve human intervention (e.g., for assessment of results) as it would bring more values and increase confidence in code quality. As explained by **R14** "*Acceptance testing [is manual] due to high frequent UI changes that is more efficient for manual testing to validate user experience against requirements*". In this regard, the survey's participants also indicated that they experienced significant burden to learning and changing mindset of customers to support fully automated acceptance testing. As stated by **R40** "*Member acceptance test [is manual]; we cannot dictate customer workflow*".

b) Tools limitations

A few of the survey participants reported that current tools and technologies are immature to fully automate acceptance tests (e.g., "*Acceptance testing [is manual], because we have lacked the tools and technology to suitably automate this*" ***R2*** or "*The tools we use [for automating acceptance tests] are too crude*" ***R91***).

c) Lack of automation skills

Lack of automation skills was another reason for having manual acceptance testing. **R2** explained the reason to adopt manual acceptance test as "*The staff involved in acceptance test phase are often domain experts with little or no automation knowledge or development skills*".

2) Manual Quality Check

The analysis of the qualitative data also indicates that although automation is critical in CD practice, manual tasks are sometimes unavoidable. For example, the organization that P15 helped to adopt DevOps was always able to deploy working versions to lower environments (e.g., staging environment) in an hour, yet the step for getting that version from low level environment into production involved additional verification and approval. That is why it was not truly CD, but it was more CDE. The most common manual task mentioned by the interviewees was code review. The interviewees' organizations performed several manual code reviews before deploying software to production. This is partially because some organizations may not have highly skilled developers for truly practicing CD. As explained by **P10:**

"*That is second code check because first the code reviews done within team and there is final check by release manager who is one of the most knowledgeable developers in the organization*" ***P10***.

Ideally, Quality Assurance (QA) tasks should be automated and integrated into CDP [36]. Several interviews' participants expressed that using automated testing could have enabled them to effectively deal with QA team and their tasks as they are not a severe bottleneck for practicing CD anymore. However, for some of them it does not mean removing all manual QA tasks from CDP. One of the interviewees reported on the reason that changes are not immediately pushed to production in the following words:

"*We fully deploy automatically on the test systems. But currently there is still a second button, required to deploy it out to the customer side. It is just because of sort of caution around, basically it would not be hard for us to automate but there is just too much concern around quality level and having another round of sign off. That is probably our biggest trouble like I said before how to get the initial quality to a level that they can be deployed simply with developer independently*" ***P7***.

Many interviewees' organizations still need to perform many manual quality checks before each release. Team members still need to release software (changes) to QA team to get certified. The QA team basically needs to manually check and confirm the level of software quality in order to being deployed to production.

According to Table I, the second most mentioned pain by the survey participants (i.e., 42.8%) for *frequent* and *automatic* deployment was conducting manual check and confirming the changes before each release. Some of the survey participants also revealed that their organizations are not willing to automate all QA tasks in CDP (e.g., due to lack of trust in existing tools) and believe that manual quality checking brings much more values to them. As one survey participant stated "*Deployments to production need a human trigger. We feel [it is] safer to look closely to the metrics during the deployment process*" ***R89***.

3) Deployment as Business Decision

Some of the interviewees indicated that in their organizations the development teams were not able to immediately deploy every change to production environment, despite passing all the tests and quality checks. This is mainly because deploying to production was considered as a business decision, which had to be made by management or financial sectors. In other words, development team members have little control over production deployment. Furthermore, different organizations may adopt different policies and timeslots for release, which can bring the most value to their customers [3, 37]. As can be seen from the following quote, despite developers could deploy changes into lower environments at any point of time, production deployment occurred every three weeks through a formal process.

"*At any point of time, [if] we wanted we could deploy into lower environments but major release was done every three weeks because the release process still was quite important for management to have sign off from the testers …" **P15**.*

We asked the survey participants to determine whether this factor (i.e., deployment as business decision) impacted their capability to automatically and continuously deploy the changes to production. According to the survey responses (see Table I), this factor was ranked as third most common confounding factor in this regard, in which out of 98 survey participants, 41 (41.8%) indicated that production deployment is considered as a business decision and is out of developers' control.

4) Insufficient level of automated test coverage

Some interviewees perceived lack of sufficient automated test coverage as a bottleneck to transition from CDE to CD (e.g., "*Of really large problem was maintaining our automated test coverage*" ***P17***). It is important to note that insufficient level of automated test coverage reduced the confidence of some interviewees' organizations in the readiness of their applications for deployment. Our survey results also confirm this concern as a large number of the survey respondents (34 out of 98, 34.7%) indicated this as a CD challenge.

5) Highly bureaucratic deployment process

We found that deployment process in some organizations is still highly bureaucratic. Our findings characterize a highly bureaucratic process as the one having a large number of formal tasks (e.g., getting approvals from various people) to be performed manually before each release. As one interviewee and one survey participant highlighted this in the following quotes:

"*Solution deployment to customer side involves taking agreement and acceptance from their backend teams…. So we were given some slots based on their delivery cycle or their priorities and we had to obey with those slots*" ***P5***.

"*Telecom operators have deployment processes that have formal bureaucratic check lists prior to deployment due to multiple integrations in their network. Solutions come from various vendors and having CD to make eco-systems work across the vendors involve manual checks*" ***R67***

Table I reveals that 31 of the survey respondents (31.6%) chose that deployment process in their respective organizations or client organizations was highly bureaucratic.

6) Lack of Efficient Rollback Mechanism

In comparison with CDE practice, CD requires better monitoring solutions and fully automated rollback mechanisms [7]. Whilst **P17** stated that integrating automated rollback in CDP increased their confidence to have automatic deployment, our analysis of qualitative data reveals that lack of having efficient rollback mechanism, forced a couple of the interviewees' organizations to decrease the pace of pushing changes to production. If there is not a very good rollback mechanism and something goes wrong in deployment process, software organizations may be at risk of delivering their customers buggy code. The results from the survey show that 24.5% of the participants confirmed that lack of efficient and automated rollback mechanism to quickly recover issues in deployment process was a reason for having manual deployment. For example, one survey participant, **R43**, stated: "*Deployment [is manual], it doesn't support rollback neither has power to provision of a machine/instance and the roll out could be better (test the released in production, load tests, micro benchmark, etc.)*" ***R43***.

7) Dependency at Application Level

Our study has found that albeit an application might be at deployable state, dependencies between that application and other systems may have inhibited some of the participants' organizations to transition from CDE to CD. It means organizations need to ensure that there is no integration problem when deploying an application to production. Deploying software changes on a continuous basis necessitates continuously deploying all dependencies (e.g., dependent applications). One interviewee described this situation vividly:

"*The difficulties become visible when you automate deployment for complex stack… Then you have sometime challenges to get everything working within one task… To all dependencies of your deployment, if [you] have legacy applications, then you need to deploy all these things together and everything should work after deployment and you always face one or two things more difficult. So simple automation you do it quite quickly and there are always some tasks to automate*" ***P9***.

As shown in Table I, the survey results moderately confirmed our interviews' findings as 23.5% of participants agreed that dependency at application level was a confounding factor. To give an example, one survey participant discussed the reason of manually deploying their application in these words:

"*Deployment is a manual process because once an artifact is created, [in order] to allow coordination with other services; dependencies must be known in advance and aren't written down*" ***R41***.

8) Demotivated customer

Based on our analysis and the interviews' discussions, we perceived that the time and pace of deployment to production greatly depends customers' cultures, polices and goals. We found that not all customers are mature enough to accept a continuous release. A number of the interviewees pointed out that whilst they were able to give updates as frequently as possible to their client organizations, the established cultures and policies in the client organizations did not support fully transition to CD practice. Therefore, they had to follow pre-defined timeslots (i.e., calendar-based release) for releasing software.

Our survey participants confirmed this finding as 19 of them indicated that their customers were not happy or had no need to receive *continuous* and *automatic* release. Two survey participants explained:

"*Upgrading to a new release is expensive and strategic for customers, they don't want to run the risks of a continuous daily or weekly delivery, they want to upgrade once a year at most*" ***R78***.

"*Moving into production [is manual] because that is not done too often; and it is a handover to Ops*" ***R13***.

Our study shows that compared to CDE, customers need to be actively involved in *continuous* and *automatic* deployment for truly practicing CD (e.g., "*I think our domain and customer are not yet in position to have continuous deployment*" ***R23***).

9) Customer environments

Our findings have revealed that the participants often found themselves struggling with customer's environment as a severe roadblock to CD. It was often stated by the interviewees that lack of carefully studying and exploring customer environments before moving to CD led to challenges in *continuous* and *automatic* release. One CDE/CD consultant observed:

"*I saw a customer actually who did not take regulation compliance into consideration and invested huge amount of time and money on doing microservices and fancy tools and at the end of the way they couldn't deliver more often than once in a month, because of regulations, ... that is the rule and that's the government stuff……Many things should be taken into considerations for success of this journey [CD practice]*" ***P14***.

Table I displays that 16 out of 98 survey respondents indicated that the challenges (e.g., manual configuration) associated to customer environment negatively affected their capability to automatically and frequently release software (changes). During our interviews and survey studies, we heard the following challenges associated with production environments as confounders to *continuous* and *automatic* deployment.

a) Manual configuration of complex software

Anecdotally, several of the interviews' and survey participants have said that manual configuration of complex software, particularly when there is a tight coupling between software and hardware, and regulatory environments represented a significant obstacle to CD success. Here are just a few of the examples indicating the participants struggled with manual configuration:

"*On the customer/onshore side, requirements developed and additional expertise was brought in to manage and support the manual delivery due to firewall and legacy processes preventing CD on the client side*" ***P3***.

"*This [production deployment] still involves a manual step to move the release artifact from one environment to another, due to network separation*" ***R71***.

"*Configuration and provisioning varies from site to site at customer location. Hence this involves manual configuration from team*" ***R67***.

Besides the complex and error-prone process of configuration and provisioning in some production environments, we also noticed several other reasons for having manual configuration and provisioning: (i) lack of mature tools (e.g., "*We do not have a robust toolset for automating configuration*" ***R49*** or "*Configuration is [manual] in Puppet but requires restarting of applications to bootstrap and load new configurations adding a manual step into the process. We use load balancing so Puppet can't restart at will*" ***R41***); and (ii) not much value in automating configuration and provisioning (e.g., "*Provisioning [is manual] because we update instance images only a few times per week*" ***R52***).

b) Hard to simulate/access real production

Our analysis highlights that a lack of control on, access, and simulation of production environment (e.g., on-premise environment) make it difficult to deploy potentially releasable software on a continuous basis. When there is no direct and regular access to customer environments, a software development team needs more communication with operations team at customer side to get confirmation and agreement for each release. The following quote depicts this issue:

"*We had a project and we had concrete control through the infrastructure and choice of technologies. That kind of environment is pretty simple to kind of get deployment pipeline that you want, you have tools, and you have kind of practices. But when we are working with customer that is not used to that model of working, that's needed to have a lot of communication, a lot of mentorship and why we need to do this things*" ***P12***.

Several interviewees shared that it is not easy (if possible) to stimulate production-like environments with realistic data. Therefore, lack of access to and control on production environment make it much harder to fully automate deployment process. One interviewee described this situation perfectly:

"*There is always challenge how to keep testing environment in synchronization with production environment. Because you can't have the same testing environment like production environment, for example network is different*" ***P14***.

Due to the above-mentioned issues, there was some debate about the real benefits of staging environment in software release process. According to the interviewees, this is mainly because staging environments do not show how really software works as a few of them explicitly stated that staging environment can be a disruptive to successfully adopt CD [20]. One interviewee told us:

"*We used to have staging environment because the reason that we needed to have a place where we integrate all the changes in one version, we test and then we deploy. But I think when we are going to more rapid deployment, all the time continuously deploy, then we started to feel it [staging] doesn't fit to this pattern because the problem is that you don't have the whole data, you don't have amount of users in staging. So, you have the risk when you deploy something in production because staging and production they are not equal.... So it is like cheating you; ... the real life is different*" ***P16***.

10) Domain Constraints

During the interviews, we found that domain constraint was a significant factor that had influenced the applicability of CD practice. This factor could change the frequency of releasing software (changes) to production [15]. Whilst CD practice is more easily applied to web-based applications, it may not be easily applied to other domains such as embedded systems and financial systems [9]. One possible reason for this is that such domains are more conservative to automatic deployment to customers and require more (manual) verification before each release [10, 14]. Table I indicates that 15 out of 98 survey participants confirmed that domain constraints do not allow them to accelerate release cadence. Whilst the survey participant **R40** told us that they are a financial exchange and are not able to deploy during business hours, another survey participant described the domain constraints in the following words:

"*I deal with Big Data style petabyte state. Large scale state migration during deployment remains a challenge at this scale due to the cost of backup and impact of lost state*" ***R46***.

11) Manual interpretation of test results

Long running tests and test results' interpretation were found as other confounding factors. Long running tests not only increased the cycle time (i.e., the time required to get code from code repository into production) in CDP, but also have hindered developers to getting real-time feedback through CDP. One interviewee revealed that a large portion of cycle time had been spent on running regression tests and interpreting the tests results. Another issue mentioned by a number of the interviewees was the fact that there was very little automation for regression tests. So it involves manual efforts and takes huge cycle time. Hence, it depends on the extent the regression tests can be automated, software organizations can significantly reduce the overall cycle time in CDP. It has been mentioned by several interviewees that with the increasing number of test cases, the interpretation of test results becomes quite time-consuming and labor-intensive process. As one interviewee reflected:

"*We have some challenges to fully automate deployment process; one of the challenges is interpretation of test results. There is manual and intervention between build the test and deploy to interpret the results*" ***P6***.

Only 11 participants in our survey (see Table I) confirmed that manual interpretation of test results was a confounder in transition from CDE to CD as one of them stated "*Acceptance testing still requires manual assessment of results*" ***R24***.

All of the above-mentioned issues together may lead organizations to doubt the quality of the code not high enough to automatically deploying to production on continuous basis.

IV. Threat to Validity

Internal validity: There are factors that could have negatively impacted our data collection and analysis processes. It was possible to select the practitioners who did not have the right kind of experience and expertise for taking part in our study. To address this issue, we applied strict criteria (e.g., seeking for potential practitioners and rigorously reviewing their public profiles) for selecting participants for both parts of this study. Additionally, we added the eligibility requirements of participants at the survey preamble. In fact, all the interviewees and almost 80% of the survey participants were selected using purposive sampling. Our results may have been affected by one specific role bias (e.g., DevOps engineer). We avoided this threat by targeting the participants holding different roles in software development. An inherent threat to validity with retrospective studies is *memory bias* (i.e., when participants cannot remember all the details) [33]. We reduced this threat as much as possible by encouraging the participants to share their experiences and lessons learnt from the last projects or clients and also by sending the interview questions to the interviewees beforehand. All this enabled them to reflect and articulate the related stories. There is a risk that participants try to provide responses that a researcher would like and want (i.e., *social desirability bias*) [38]. For both studies, the participants were ensured that all the collected data would be kept anonymous under human ethics approval obtained for this study.

Data triangulation (i.e., using different data sources) was another validity approach to reduce researchers' bias. If at least two interviewees discussed one fact, we presented it as an interview's finding and used it for formulating the survey questions. This strategy helped us to alleviate negative impact of any possible subjective viewpoint of the results. Whilst the first author has mainly performed qualitative data analysis and interpretation, we adopted the following method to mitigate researchers' bias in this regard: the coded data was evaluated and verified by other authors through frequent meetings.

Construct validity: Appropriateness and comprehensibility of the questions and answer options used in both the interviews and the survey can be another source of threat in our study. In order to deal with this threat, the first author designed the interviews' questions based on a systematic review [39] and multi-vocal review, with seeking feedback and validation from the other authors and a few industrial practitioners. The feedback collected at the end of the interviews and the survey was valuable as it helped us to fine-tune some questions (e.g., changing questions wording) that were confusing or unclear. Moreover, the exact definitions of CDE and CD may differ between practitioners. To reduce this threat, we carefully defined these terms to our participants as none of them showed disagreement. Whilst the interviews' findings led to the creation of the survey questions, wherever required we also allowed the participants to reflect more thoughts and perspectives by including open-ended questions or "Other" field in the survey questions responses. Thus, we are confident that our questions covered the important confounding factors in moving from CDE to CD.

External validity: In order to increase the generalizability of our results as much as possible, we attempted to purposefully target as diverse a population as possible in regard to roles, experiences, expertise, organizations' size, domain, and geographical locations. Our sampling technique gives us some confidence that our results to a large extent are representative. Additionally, the interviews' findings were evaluated and generalized by 98 survey respondents.

V. Discussion

This section first discusses some of the main findings from our study. Second, we suggest implications for practitioners and researchers based on the themes that emerged from five-top reported factors and, analysis of the responses to an open-ended question: "*Given the increasing importance of automation in CD, in your understanding what are the top four things that you look for/need/would like to see in automation*".

1) Summary of main findings

Our study indicates that there ***is*** a well-understood difference between practicing Continuous DElivery (CDE) and Continuous Deployment (CD) in software industry. Specifically, there are factors because of which organizations may be unable or demotivated to move from CDE to CD (e.g., having *automatic* and *continuous* deployment). These factors are "lack of fully automated (user) acceptance test", "manual quality check", "deployment as business decision", "insufficient level of automated test coverage", "highly bureaucratic deployment process", "lack of efficient rollback mechanism", "dependency at application level", "demotivated customer", "customer environment", "domain constraints", and "manual interpretation of test results". Moreover, we found that most of the participants' organizations still have semi-automated CDPs, in which "acceptance testing", "production deployment", and "configuration and provisioning" stages have least automation support.

2) Implications for research and practice

Better automated testing: Both the interview and survey data show a strong need of better support for automated testing, in particular (user) acceptance testing. Several of the participants mentioned that the current automated testing tools need significant improvements in order to harden them for different environments. The participants thought that (user) acceptance testing on relative scale have to be run and assessed by a human as it brings more value and safety. The participants also mentioned the need to test all types of applications (for example mobile testing as it is fragile and expensive to automate), techniques and tools that enable parallelization of automated testing and infrastructure automation testing.

Integrating automated quality checks: Software quality was one of the major concerns in the interviewees' and the survey participants' organizations before each release. It was also among the top priorities for business leaders. Security and performance were the most frequently reported quality concerns. According to the participants, attention to security needs to increase and performance testing should be conducted at production scale. However, an open question is how to efficiently automate quality checks inclusive of performance and security and incorporate them into CDP. For instance, there is a strong need of integrating performance baselines into CDP.

Management support: Participants perceive "managers" are hesitant to allow developers immediately push out every change to production as only business leaders of their organizations are responsible to make decisions about when and what to be deployed to production. Our analysis shows that compared to CDE, successfully adopting CD needs better management support. This is mainly because deployment on a continuous basis without human intervention may increase complexity as organizations need to deal with more components, more people, more roles, and more concerns. Hence, this can be much more a business problem or a political problem rather than just an engineering problem. Management at both customer and vendor organizations is expected to have a clear understanding of business drivers of continuous deployment, and get all the stakeholders on board. Whilst the main business leaders' concern is around quality level, our results suggest that integrating automated quality checks and security test in both development and operations processes can alleviate this concern and to a large extent make continuous deployment compelling to business leaders. To achieve CD, organizations must break down barriers at production. This is mainly achievable by allowing developers to be part of deployment decision and more trusting them. By this, the manual approval process described in Section IV.C.5 will be significantly reduced.

Easier tools integration: As we discussed in Section IV.B, CDP is a tool-chain, which a number of open source and/or commercial tools should be integrated for this purpose. This is mainly because deploying with *one and only tool* would make automatic deployment process more complicated. However, a commonly mentioned issue was compose-ability of tools. Software organizations need to spend too much engineering effort to architect each of distinct tools to interface and integrate with other tools to make them work seamlessly. We observed that due to availability of a gamut of tools and lack of standardization between them, there is too much of chaos in the way each organization adopts their continuous delivery or deployment journey. It is highly recommended that tool vendors consider applying standards that enable an organization to easily stitch tools together. Such standards would drastically minimize the difficulty and effort required for tools integration.

Digestible visualization and monitoring: Although there are lots of monitoring tools available, the survey participants often expect tools and techniques, which enable them to have full monitoring coverage. In the meanwhile, having a visual representation of end-to-end build, test and deployment would make a huge difference in deployment capability of organizations to release faster and often. Unfortunately current tools are not great for this; presumably because they do not do a good job (e.g., lack of domain specific monitoring tools) or are exceedingly complex for the job. Furthermore, scaling CD practice in large organizations with multiple teams and applications can worsen this problem as a wide range of stakeholders are needed to be able to understand what is happening, what has happened, and why in a real time manner. Hence, there is a need to develop tools that provide real-time, digestible and customizable monitoring and alerting for different type of stakeholders [40, 41].

Other needs: There are also serious needs for (1) better tools to simplify configuration and provisioning of environments and support automatic setup of distributed environments; (2) tools to manage, validate and automate schemas upgrades and database migrations in CDP; and (3) better post deployment checks (e.g., automated smoke and reliability testing after deployment.

VI. Related Work

This section places our work in the context of other related studies. Lwakatare et al. [14] present high level challenges for the adoption of DevOps in embedded systems domain. The identified challenges are huge dependency between hardware and software versions, lack of access to customer environments, and lack of appropriate technologies to automatically and continuously deploy software to customer environments. Whilst [11, 12] present the obstacles and challenges to adopt CDE practice, adopting CD practice has been evaluated by [15-17]. Claps et al. [16] report the challenges that a single case software company faced in transition to CD. The identified challenges are classified into technical and social ones including team experience, continuous integration, partner plugins, and changing database schemas. The study also reveals what solutions (e.g., adopting social rules and investment in infrastructures) the case company employed to address those challenges.

Savor et al. [17] investigate the effect of adopting CD practice on team productivity (i.e., number of added or modified code lines pushed to production per developer) and product quality (i.e., number of failures in production) at Facebook and OANDA. The study reveals that whilst the number of team members and complexity of code size drastically increased over six years, practicing CD had no negative impact on team productivity and quality. The study discusses a number of challenges including management support and extra effort for understanding updates that an organization may face in adopting CD. The challenges identified for adopting CD [11, 12] are almost similar to those that are found for CDE [15-17]. For example, most of the studies indicate that manual testing, unsuitable architecture, and resistance to change are roadblocks to practicing CDE and CD. In addition to the challenges reported in [12, 15-17], Laukkanen et al. [13] show that stage-gate process negatively impacts on CDE success. This is mainly because the attributes (i.e., tight schedule) of stage-gate process significantly limit the time needed for CDE adoption. The study argues that it is almost impossible to adopt CDE in a stage-gate managed organization without changing the process. Most of the previous studies *did* consider CDE and CD as one practice and did not distinguish the challenges associated with adopting CDE and CD. To the best of our knowledge, our paper reports the first (large scale) piece of work, which distinguishes CDE from CD and empirically investigates the confounding factors that limit or demotivate organizations from moving towards CD from CDE.

VII. Conclusion

This paper has reported an empirical investigation into the reasons (e.g., manual user acceptance testing) because of which organizations may be unable or demotivated to *automatically* push out every change to production in order to have many production deployments every day. Our findings came from a mixed-method study consisting of data collection and analysis from 21 semi-structured interviews and an online survey completed by 98 software practitioners. This research reveals the current state of automation support in software industry to truly implement continuous deployment. Interestingly, the majority of the participants' organizations did not have fully automated deployment pipelines, with mostly semi-automated or manual "acceptance testing", "production deployment", and "configuration and provisioning" stages. We have also identified several future research directions (e.g., better tooling support) along with a set of recommendations (e.g., management support) that can help streamline *continuous* and *automatic* deployment.

ACKNOWLEDGMENT

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 supported by Australian Government Research Training Program Scholarship.

REFERENCES

[1] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*: Addison-Wesley Professional, 2015.

[2] *2015 State of DevOps Report. Available at: https://puppetlabs.com/2015-devops-report [Last accessed: 5 October 2015]*. 2015.

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

[4] M. Fowler. "Continuous Delivery. Available at: http://martinfowler.com/bliki/ContinuousDelivery.html [Last accessed: 21 October 2015]." 21/10/2015; http://martinfowler.com/bliki/ContinuousDelivery.html.

[5] B. Fitzgerald, and K.-J. Stol, "Continuous Software Engineering: A Roadmap and Agenda," *Journal of Systems and Software,* vol. 123, 2017.

[6] E. Luke, and S. Prince. "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]."

[7] I. Weber, S. Nepal, and L. Zhu, "Developing Dependable and Secure Cloud Applications," *IEEE Internet Computing,* vol. 20, no. 3, pp. 74-79, 2016.

[8] T. Dingsøyr, and C. Lassenius, "Emerging themes in agile software development: Introduction to the special section on continuous value delivery," *Information and Software Technology,* vol. 77, pp. 56-60, 2016.

[9] M. Skelton, and C. O'Dell, *Continuous Delivery with Windows and .NET*: O'Reilly 2016.

[10] H. H. Olsson, H. Alahyari, and J. Bosch, "Climbing the "Stairway to Heaven" -- A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software.," in 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 2012, pp. 392-399.

[11] S. Neely, and S. Stolt, "Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy)," in Agile Conference (AGILE), 2013, pp. 121-128.

[12] C. Lianping, "Continuous Delivery: Huge Benefits, but Challenges Too," *IEEE Software,* vol. 32, no. 2, pp. 50-54, 2015.

[13] E. Laukkanen, T. O. A. Lehtinen, J. Itkonen, M. Paasivaara, and C. Lassenius, "Bottom-up Adoption of Continuous Delivery in a Stage-Gate Managed Software Organization," in 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Ciudad Real, Spain, 2016, pp. 1-10.

[14] L. E. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. H. Olsson, J. Bosch, and M. Oivo, "Towards DevOps in the Embedded Systems Domain: Why is It So Hard?," *49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 5437-5446.

[15] M. Leppanen, S. Makinen, M. Pagels, V.-P. Eloranta, J. Itkonen, M. V. Mantyla, and T. Mannisto, "The Highways and Country Roads to Continuous Deployment," *IEEE Software,* vol. 32, no. 2, pp. 64-72, 2015.

[16] G. G. Claps, R. Berntsson Svensson, and A. Aurum, "On the journey to continuous deployment: Technical and social challenges along the way," *Information and Software Technology,* vol. 57, pp. 21-31, 2015.

[17] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at Facebook and OANDA," in 38th International Conference on Software Engineering Companion, Austin, Texas, 2016, pp. 21-30.

[18] E. Laukkanen, J. Itkonen, and C. Lassenius, "Problems, causes and solutions when adopting continuous delivery—A systematic literature review," *Information and Software Technology,* vol. 82, pp. 55-79, 2//, 2017.

[19] S. G. Yaman, T. Sauvola, L. Riungu-Kalliosaari, L. Hokkanen, P. Kuvaja, M. Oivo, and T. Männistö, "Customer Involvement in Continuous Deployment: A Systematic Literature Review," *Requirements Engineering: Foundation for Software Quality: 22nd International Working Conference, REFSQ 2016, Gothenburg, Sweden, March 14-17, 2016, Proceedings*, M. Daneva and O. Pastor, eds., pp. 249-265, Cham: Springer International Publishing, 2016.

[20] T. Fitz. "Continuous Deployment: Beyond Continuous Delivery. Available at: http://www.slideshare.net/TimothyFitz/continuous-deployment-beyond-continuous-delivery [Last accessed: 21 December 2016].".

[21] L. W. Richter. "Getting from Continuous Delivery to Continuous Deploymenty. Available at: https://www.youtube.com/watch?v=PFFifnIn348 [Last accessed: 18 December 2016].".

[22] K. Lankford. "Beyond Continuous Delivery—All the Way to Continuous Deployment. Available at: https://www.stickyminds.com/presentation/beyond-continuous-delivery-all-way-continuous-deployment [Last accessed: 13 January 2017].".

[23] S. Mäkinen, M. Leppänen, T. Kilamo, A.-L. Mattila, E. Laukkanen, M. Pagels, and T. Männistö, "Improving the delivery cycle: A multiple-case study of the toolchains in Finnish software intensive enterprises," *Information and Software Technology,* vol. 80, pp. 175-194, 2016.

[24] G. Schermann, J. Cito, P. Leitner, and H. C. Gall, "Towards quality gates in continuous delivery and deployment." pp. 1-4.

[25] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting Empirical Methods for Software Engineering Research," *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer and D. I. K. Sjøberg, eds., pp. 285-311, London: Springer London, 2008.

[26] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case studies for method and tool evaluation," *IEEE Software,* vol. 12, no. 4, pp. 52-62, 1995.

[27] S. E. Hove, and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research." pp. 23-32.

[28] L. A. Palinkas, S. M. Horwitz, C. A. Green, J. P. Wisdom, N. Duan, and K. Hoagwood, "Purposeful Sampling for Qualitative Data Collection and Analysis in Mixed Method Implementation Research," *Administration and Policy in Mental Health and Mental Health Services Research,* vol. 42, no. 5, pp. 533-544, 2015.

[29] L. A. Goodman, "Snowball Sampling," *Annals of Mathematical Statistics,* vol. 32, no. 1, pp. 148-170, 1961.

[30] D. S. Cruzes, and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM), 2011, pp. 275-284.

[31] B. A. Kitchenham, and S. L. Pfleeger, "Personal Opinion Surveys," *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer and D. I. K. Sjøberg, eds., pp. 63-92, London: Springer London, 2008.

[32] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: the contributor's perspective," in 38th International Conference on Software Engineering, Austin, Texas, 2016, pp. 285-296.

[33] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The Design Space of Bug Fixes and How Developers Navigate It," *IEEE Transactions on Software Engineering,* vol. 41, no. 1, pp. 65-81, 2015.

[34] B. Adams, and S. McIntosh, "Modern Release Engineering in a Nutshell -- Why Researchers Should Care," in IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016, pp. 78-90.

[35] A. Phillips, M. Sens, A. de Jonge, and M. van Holsteijn, *The IT Manager's Guide to Continuous Delivery: Delivering business value in hours, not months*: XebiaLabs, 2015.

[36] *2016 State of DevOps Report. Available at: https://puppet.com/resources/whitepaper/2016-state-of-devops-report [Last accessed: 15 March 2016]*. 2016.

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

[38] F. Adrian, "Response bias, social desirability and dissimulation," *Personality and Individual Differences,* vol. 7, no. 3, pp. 385-400, 1996.

[39] M. Shahin, M. A. Babar, and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," *IEEE Access,* vol. PP, no. 99, pp. 1-1, 2017.

[40] M. Brandtner, E. Giger, and H. Gall, “SQA-Mashup: A mashup framework for continuous integration,” *Information and Software Technology,* vol. 65, pp. 97-113, 9//, 2015.

[41] D. Ståhl, K. Hallén, and J. Bosch, “Continuous Integration and Delivery Traceability in Industry: Needs and Practices,” in 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2016, pp. 68-72.