Communication of Changes in Continuous Software Development

Telcio Elui Cardoso(✉), Alan R. Santos, Rafael Chanin, and Afonso Sales

School of Technology, PUCRS, Porto Alegre, RS 90619-900, Brazil
telcio.cardoso@edu.pucrs.br, alan.ricardo@acad.pucrs.br, {rafael.chanin,afonso.sales}@pucrs.br

Abstract. The industry competition has changed the way software is managed, developed, and delivered over the years. Some of the approaches that emerged to continuously deliver software to users are Continuous Delivery (CDE) and Continuous Deployment (CD). CDE is the ability to get software functionalities, of any kind, into the hands of users, in small batches, and short cycles. In CD, every change that passes all stages of the production pipeline is released to customers without human intervention. The agility proposed by the Continuous Delivery and Continuous Deployment approaches may introduce some challenges to the software development life-cycle. Some of these challenges are related to the Software Configuration Management process and the communication of software changes to relevant stakeholders such as operations teams. In order to better understand which communication practices are used to communicate software changes in environments where Continuous Delivery or Continuous Deployment were adopted, a systematic literature review and an empirical study with global companies were performed, which allowed us to consolidate a collection of communication practices for communicating software changes that could benefit companies that already adopted or are planning to implement continuous software delivery practices.

Keywords: Software engineering · Software development · Continuous integration · Continuous delivery · Continuous deployment · Continuous release · DevOps

1 Introduction

The growing demand for faster time to market cycles has shaped the way software products were managed, developed and delivered to users over the years. In order to support the faster pace required by the market, *Continuous Software Development* (ConSD) practices such as *Continuous Delivery* and *Continuous Deployment* have emerged in the recent years. A software release may introduce changes or new functionalities to an existing application, which may

E. Klotins and K. Wnuk (Eds.): ICSOB 2020, LNBIP 407, pp. 86–101, 2021.
https://doi.org/10.1007/978-3-030-67292-8_7

require specific communication to help Customer Support Services teams to better understand the changes introduced. Therefore, understanding how the speed of software changes imposed by ConSD practices impact the communication of software changes is extremely important to ensure Customer Support Services teams will be able to support end-users properly.

In order to understand this phenomena, this study performed a Systematic Literature Review (SLR), followed by an empirical study with seven global companies which then had the results triangulated and consolidated in a set of communication practices.

2 Methodology

In order to understand the challenges related to communication in continuous delivery and continuous deployment environments, a systematic literature review (SLR) has been performed, following Kitchenham [17] guidelines. Moreover, in order to get an in-depth understanding about the communication practices in use in the software industry and complement the systematic literature review results, we performed an empirical study with seven participants from seven different companies.

3 Literature Review

According to Kitchenham and Charters [4] a systematic review protocol is a plan that describes the conduct of a proposed systematic literature review. In the following paragraphs we describe the elements that compose the systematic review protocol of this research.

Research questions are used to drive a systematic review process. Based on these questions, a systematic search is performed to identify relevant information, such as papers and journals, which could contribute to the goal of this study. Following, the questions we have defined in our research are presented:

- **RQ1**: How do ConSD practices impact the communication of software changes?
- **RQ2**: How do software changes are communicated to customer support services teams in environments where ConSD practices were adopted?

A good way to create a search string is to structure them in terms of population, intervention, comparison, and outcome [17]. The search string used in this study is presented in Table 1. Additionally, Table 2 describes the inclusion and exclusion criteria applied to our research.

3.1 Data Extraction

Following a systematic literature review protocol, the research questions were used to search for relevant information on the ACM, IEEE, Scopus online

Table 1. Search string

Type	String
Population	(Software engineering OR software development AND)
Intervention	(Continuous deployment OR continuous delivery OR continuous integration OR continuous release OR devops)

Table 2. Exclusion/Inclusion criteria

Type	Description
Exclusion	Event keynotes, summaries, extended abstracts
Exclusion	Papers with less than 4 pages
Exclusion	Language different than English
Inclusion	Papers published between 2010 and 2019
Inclusion	Papers where CI, CDE or CD practices were mentioned

databases. Our initial search results returned 5,348 studies, which were filtered resulting on 102 full papers read. After reading carefully 102 papers, 39 papers were selected based on their relevance. The results of our systematic literature review process are presented in Sect. 3.2.

3.2 Results

In the following paragraphs, this study answers the research questions initially formulated as part of the research protocol and provides details about the challenges and practices identified.

(RQ1) How do ConSD practices impact the communication of software changes?

In the study conducted by Klepper *et al.* [18] the authors argue that many releases can overwhelm users, especially if they do not understand the difference between versions being released. The authors still argue that release notes can help in such situations, however, creating high quality release notes, manually, requires expressive amount of time and effort.

Shahin *et al.* [36] argue that a better visualisation of the CDE process would make a huge difference in the capability of the organisations to release faster and often. In one of the studies described by Shahin *et al.* [35], the authors argue that the status of a project should be visible and transparent to all team members. Shahin *et al.* [35] also argue that coordination and collaboration are challenges for continuous practices. According to their study, as more frequent software is deployed, more communication and coordination with operations teams is required. Their study also describes issues related to merge conflicts caused by the lack of awareness around software changes.

Brandtner *et al.* [5] argue that one of the main issues related to CI environments is the fact that relevant information about the health and quality of the software is spread across several tools and multiple views.

Stahl *et al.* [42] argue that traceability is a key challenge in achieving CI and CDE and describe an industry developed framework named Eiffel, designed to provide real time traceability for ConSD environments. The authors argue that agile methodologies, CI and CDE might be challenging for traceability due to the need of overhead reduction and traceability requirements. Stahl *et al.* [41] argue that even though the fundamental aspects of traceability remain the same regardless of CI and CDE, the nature of faster releases, increased frequencies have increased the amount of data generated in these processes, created new challenges in practice. Palihawadana *et al.* [28] argue that traceability links can be used by DevOps teams to evaluate software changes and their impact, however, also argue that maintaining traceability in agile based environments has become a challenge due to the lack of proper management tools and poor documentation.

Lwakatare *et al.* [25] highlighted studies which described the negative impact on project releases, from time to quality, due the lack of poor communication and lack of early involvement of operations teams in the software development process.

Yaman *et al.* [46] argue that one of the challenges related to communication in CD environments might be the excess of transparency that may lead to customers interference in developers' work.

Olsson *et al.* [27] argue that some of the challenges in such environments are the communication and coordination with suppliers, difficulties of getting an overview of the status of the project and the lack of transparency, including information available to users and other stakeholders who might need them.

Alyahya *et al.* [2] describes some challenges that affect the development progress and highlight the difficulties related to communication in distributed teams in order to maintain an awareness about development progress if they rely just on ad-hoc communication of changes. Alyahya *et al.* [3] argue that it is difficult to keep team members from different sites aware about each one's work.

Diel *et al.* [10] argue that some of the challenges faced in such environments are the lack of training on applications changes and no previous notice of software releases.

Downs *et al.* [11] argue that software teams working in agile projects produce a great deal of information on a daily basis, however, these information are misapplied, communicated ineffectively or ignored, which has impact in the results of these projects. In the study conducted by Shahin *et al.* [35], several papers described the lack of team awareness and transparency among team members as a challenge that may break down transition towards continuous practices. Table 3 has been created to group challenges categories and sumarize the challenges around software changes, previously described. Additionally, Table 4 categorizes papers and their respective communication challenge category.

Table 3. Communication challenges categories

Category	Description
Discovery	Information regarding software changes are not available, are not easily accessible or are not properly communicated
Coordination	Software changes require activities coordination between different teams such as software development and operations teams
Traceability	Tracing software changes from an end-to-end perspective considering the amount of data generated by CI and CDE environments
Training	Training teams to be up-to-date regarding software changes represent a challenge in in high frequency changing environments

Table 4. Papers per category

Challenges	Studies
Discovery	[2,3,5,8,10,11,18,25,27,32,35,36,46]
Coordination	[35]
Traceability	[28,41–43]
Training	[10]

(RQ2) In environments where Continuous Delivery practices were adopted, how software changes are communicated to customer support services team?

In the studies [25,31], the usage of shared Kanban boards has been described as a practice to communicate software changes. The usage of shared Dashboards has been described as a practice in the studies [11,16,29,34,38]. The usage of Ambient Surfaces and Project Radiators have been described by the studies [31,33,44] as practices to ensure daily progress on projects is completely transparent and available for all stakeholders. Punjabi *et al.* [29], also argue that *user stories* can be maintained and assigned in such a system and could be integrated with source control and CI server to obtain issues addressed in a commit or build.

Centralizing and exchanging changes information through Tracking Systems has been described by the studies [1,2,12–14,19,32,33,39,40,43]. Downs *et al.* [11] and Kim *et al.* [16] mention the usage of issue tracking information along with dashboards and information radiators in order to provide project status. Punjabi and Bajaj [29] describe the usage of bug/issue/task tracking features along with visual boards that support agile development.

In the study conducted by Krusche *et al.* [19], the authors relate the usage of release notes, collected automatically by the CI server and linked to issues in tracking systems to provide visibility about the changes included in a given software release. Klepper *et al.* [18] propose a solution that uses a semi-automatic approach to release notes generation to reduce workload for the development team and release manager while still allowing them to provide properly targeted

content depending on the context and recipients of a release. Still according to the authors, the information that is already produced during the development process is used to prepare release notes content. Kula *et al.* [20] relate the impact of missing release notes on rapid-releases with are delayed in such cases.

In the study performed by Neely *et al.* [26], the authors described some process changes required to implement CDE, which included the replacement of big meetings by crisp emails along with internal wikis and blogs in order to share knowledge about new features with other teams such as sales and support teams. Younas *et al.* [47] describe the usage of email, among other tools for collaboration in agile development. Brandtner *et al.* [6] describe the usage of email as a notification channel for CI-Tools. Krusche *et al.* [19] describe the usage of email to send release notes to users with details about solved issues, which can be also checked through an issue tracking system.

Wiedemann *et al.* [45] argue that traces are essential for application management, therefore, everything must be logged. In one of the studies evaluated by Shahin *et al.* [35], the authors suggest the practice of recording changes to a change log and making it visible to customers in order to enable them to track features changed. According to Shahin *et al.* [38], *et al.* [37], organizations practicing CD need to appropriately record, aggregate and analyse logs and metrics as an integral part of their CD environment in order to hypothesise and run experiments for examining different functionalities of a system. Shahin *et al.* [38] still argue that readability of logs for all stakeholders should be taken into consideration due to the fact that developers use to build the logging mechanism into the source code and there is a chance that such logs might become too technical for IT operations teams, such as support people, and may not be efficiently used to their needs. Lai *et al.* [21] describes the logging of relevant information such as programmer and change reasons information in the check-in step of CI processes in order to share this information with relevant stakeholders.

Senapathi *et al.* [34] describe the usage of Yammer to share software releases information with others stakeholders and to promote discussion on completed tasks and lessons learned and Atlassian Confluence to share release plans and software documentation. Schwarzer *et al.* [33] describe the usage of Atlassian Confluence along with ambient surfaces to share build summaries, reports and errors as well as software architects announcements. Neely *et al.* [26] describe the usage of wikis and blogs, replacing big meetings, in order to share information about new features. Heesch *et al.* [43] relate the usage of Atlassian Confluence along with Atlassian Jira to share application design definitions with team members. Heesch *et al.* [43] argue that apart from supporting the realisation of software artefacts, the information available in the tools also serve documentation needs, allowing the visualisation of tasks history. Claps *et al.* [8] describe Atlassian's case, a well established software company, where Confluence and blogs are used for software documentation and customer feature discovery purposes. Leite *et al.* [22] describe the usage of the GitLab tool as a wiki system for knowledge sharing between developers and operators.

In the study performed by Leite *et al.* [22], the authors mention the usage of ChatOps (Chat and Operations) tools in DevOps environments as a model that connects people, tools, processes, and automation through conversation-driven interactions. Lwakatare *et al.* [25] mention the HipChat tool usage for interaction between developers and operators, particularly when setting-up new environments. In the study performed by Luz *et al.* [24], the continuous use of instant messaging tools like Slack and HipChat was cited as the most appropriate option for communication between developers and operators in DevOps environments. In the study performed by Downs *et al.* [11], Instant Messaging conversation was the primary form of communication within the team, and for a wide range of purposes, including on-going status updates and team collaboration. Table 5 summarizes the main communication practices adopted by ConSD teams in order to ensure software changes are visible by other teams in the software development life cycle process.

Table 5. Communication practices

Practices	Studies
Sharing information through visual boards	[5, 6, 11, 15, 16, 23, 25, 31, 33, 34, 38, 44]
Exchanging information through tracking systems	[1, 2, 11–14, 16, 19, 22, 29, 32–34, 39, 40, 43]
Communicating changes through release notes	[18–20]
Communicating changes through Email	[6, 19, 26, 47]
Leveraging Logs to share information	[21, 32, 35, 37, 38, 45]
Communicating changes through collaborative workspaces	[8, 22, 26, 33, 34, 43]
Exchanging information through instant messaging	[11, 14, 22, 24, 25, 30, 34]

Based on these results, we performed an empirical study in the industry, which would complement the SLR results. The empirical study is presented in details in Sect. 4.

4 Empirical Study

In order to get an in-depth understanding about the communication practices in use in the ConSD industry and complement the SLR results, we performed an empirical study with seven participants from seven different companies. In the following paragraphs we provide details regarding the methods used to perform this research, from data collection and criteria to select the participants, to the data analysis method applied and finally the results obtained.

4.1 Data Collection and Analysis

Following Creswell's [9] recommendation, a questionnaire has been created to support the empirical study. In order to answer the research questions, we collected information from seven different companies. The interviewees and their

companies were anonymised due to non-disclosure agreements, where we use aliases from Company A to Company G. The participants and their companies are briefly described in Table 6.

Table 6. Companies, practices, roles and experience

Company	Employees	Approach	Role	Yrs role
Company A	3,500	CDE	Product Manager	2
Company B	6,000	CD	Performance Engineer	6
Company C	50,000	CDE	Software Architect	3
Company D	2,000	CDE	Platform Engineer	1.3
Company E	7,000	CD	Project Manager	3
Company F	500	CDE	Software Developer	9
Company G	2,000	CDE	Support Engineer	9

4.2 Results

At Company A, software development teams and customer support services teams work apart from each other. The communication between software development teams and customer support services teams, in general, occurs through instant messaging tools or through their issue tracking systems. The *Service Enablement Team* (SET) is a specific team that makes a bridge between software development teams, customer support services teams and customers, mainly responsible for ensuring major software changes will be properly managed in order to avoid any impact on customers and customer support services teams. The SET team organizes bi-weekly meetings with customer support services teams where the main upcoming product changes are communicated, specially the bigger ones. Bigger software changes are communicated to customers through blog posts and built-in modal product banners. Developers and support teams use to interact using hidden messages through their issue tracking systems in order to follow-up changes required to fix bugs or features requested by customers. Even having daily deploys of product changes to production small changes are not communicated to customer support services teams on a daily basis, which represent some challenges to the support teams who sometimes receive customers complaints about changes in the products that are not reflected to the public documentation and were not communicated to them. Company A uses Atlassian Confluence to support the communication of major changes and releases in their products to customer support services teams.

At Company B, software changes, in general, are not communicated to customer support services teams and end-users. Whenever further information is required regarding a software change, instant messaging, memo pages and emails are the main channels to share information regarding software changes between

development and customer support services teams. The usage of a change log database has been described in order to record several types of change logs, which can be accessed and tracked by the customer support services teams by using tools developed by internal teams specifically for this purpose.

At Company C, the communication of software changes, between software development and customer support services teams, occurs mainly through tickets in their issue tracking system, email messages and formal meetings. Formal meetings with key people in the customer support services teams are performed once a month to share details about the major changes being released to production, being these key people in support responsible for sharing details about the changes with their support teams. At Company C, customer support services teams have access to the development teams summarize Boards, therefore, they can also have visibility about the upcoming software changes being developed or about to be released. Release notes are used to share changes details and are available to software development teams and customer support services teams. End-users are communicated about software changes through a *"What's New"* web page, that is built into the product.

At Company D, most of the communication around software changes is performed through instant messaging and email channels. Projects use to structure their information using internal wiki pages which are then open to other teams that can track information about software changes and projects status. Issue tracking systems are also used to communicate the progress and communicate the status of software changes, specially bug fixes and feature requests.

At Company E, formal communication of software changes is mainly restricted to a showcase meeting, organised by the Product Owner. In this meeting, which use to occur every 15 days, the status of new features and bug fixes, among other project relevant information use to be shared with relevant stakeholders, such as customer support services teams and other software development teams. Additionally, the communication between different teams around software changes use to occur through instant messaging tools and tickets created at their tracking system where comments are added in order to provide details and status of such software changes.

At Company F, communication of software changes uses to be performed by Product Owners who share details about the changes with account managers and customers. There are no specific criteria to define which software changes will be communicated to the customers and is under the Product Owner the responsibility to decide which changes should be communicated and which one should not. Product Owners use to communicate software changes to customers through email and face-to-face meetings. The communication channel and frequency of these communication varies based on the requests being released.

At Company G, the communication of software changes to customer support services agents is facilitated by the fact that their software development teams include the customer support role in the team structure. End-users do not use to be notified about software changes, mainly because such changes use to have a minimum impact on the way customers use their software. Additionally, teams

use to share information about software changes through instant messaging channels and CI tools are also used as source of information to get status and further technical details about software changes. The company is working to implement an issue tracking system in order to better support the flow of requests between teams.

5 Threats to Validity

Even though we have followed a SLR protocol, some threats to validity might be identified: (i) the research strategy is not correct; and (ii) the research bias regarding some of the studies selected. Such threats to validity were minimised by having this research protocol as well as the results of this work, reviewed by other two researchers. Regarding our empirical research, even though we have followed a protocol to conduct the interviews, the number of participants of this empirical study might represent a threat to the validity. Interviewing additional participants from other companies or even within the same companies, would have been useful to complement the results of our study.

6 Discussion

This section presents the main results of this research which were obtained by triangulating the SLR and the empirical study results. The data triangulation results are discussed and presented as a set of good practices.

Good Practice 1 - *Increasing awareness and transparency around software changes by sharing information through kanban boards, dashboards and information radiators with relevant stakeholders.*

Challenges related to the communication and visualization of project status have been reported in the SLR research by the studies [6,11,27,32,35,38,42]. Shahin *et al.* [38] argues, based on the studies evaluated as part of their research, that a better visualisation of the end-to-end software development life-cycle would allow software teams to release faster and often. In this sense, the usage of shared electronic Kanban Boards between different teams allows a visual tracking of software changes and has been reported as a common practice between the interviewees in the companies A, C, E and F as well as by the studies [25,31]. Team members can navigate in the content of User Stories, in order to better understand details of each change such as feature details, planning release dates, stakeholders affected and developers responsible for such features. Such level of transparency improves communication and provides autonomy to teams, which contributes to organisations in several aspects, from risk reduction to customer satisfaction. The implementation of this practice should take into consideration that, in case of customers adoption, the level of transparency may have negative implications, once customers may interfere in the development process frequently, as stated by Yaman *et al.* [46], reducing the release speed.

The traceability of changes could take advantage of shared Dashboards, once, as electronic Kanban Boards, they provide a visual mechanism to keep other stakeholders aware about progress and status of software changes, with the advantage of allowing layout and content modelling according to information needed and the target audience.

The usage of Information Radiators has been reported in the studies [16, 31,33,44] as an effective practice to increase transparency and awareness around project status and software changes for all stakeholders. Along with Dashboards, can be a practice for those teams which work at the same physical, once rely on infrastructure aspects to be used.

Good Practice 2 - *Using tracking systems to provide information around software changes to relevant stakeholders, from bug fixes to feature requests.*

The usage of tracking systems has been reported by the studies as well as by the companies A, B, C, D, E and G. Tracking systems were described in the literature and in the industry as tools to centralize information from different sources and track the software development progress, usually represented by bug-fixes and customers' requests. Such systems allow real-time interaction between different stakeholders and use to provide lower level granularity information, mainly focusing on operational information. The information exchanged between software development teams, operations teams, users, among other stakeholders, occurs through text comments in the tickets or through notifications, which might be triggered automatically every time there is a change in the status. Tracking Systems might be also integrated to other CI tools in order to support the deploy process.

Good Practice 3 - *Logging software changes to a central log database and sharing these information with relevant stakeholders.*

The practice of logging software changes to a central change log repository and sharing this information with relevant stakeholders has been cited in the studies [32,35,37,38] and by the companies B and D. At Company B, the interviewee has described the usage of a change log database in order to record software changes performed at the company, which can be used by Customer Support Services teams to track software changes. Such information might be used for traceability purposes as well as to report issues related to the deployed functionalities. The practice of logging software changes may complement the information provided by Tracking Systems. Additionally, a central source of changes logs could be integrated with other tools to improve the communication of changes, such as Kanban Boards and Dashboards as well as be used to generate automatic notifications and release notes.

Good Practice 4 - *Automating the release notes generation process in order to reduce manual efforts.*

The usage of Release Notes have been reported by the studies [7,18–20] as well as by the companies C and G. As frequent as software development teams release changes to users, less is the usage of release notes as a mechanism to communicate changes, once they require manual intervention to be created. Therefore, the usage of Release Notes is a recommended practice, however, due to the fact that its creation requires manual effort, its usage might not be recommended to development teams where a high volume of software changes and deploys are performed on a daily basis. The usage of some level of automation to generate automatic release notes as described in the literature by Krusche *et al.* [19] and Klepper *et al.* [18] might represent an alternative to reduce the effort necessary to create release notes.

Good Practice 5 - *Using email channels to share details about software changes with relevant stakeholders.*

The usage of Email, sometimes as channel and in others as communication practice, is still recommended in ConSD environments, however, as the practice of Release Notes, its usage is directly related to the frequency of software releases deployed in production environments. As the time passes and companies move from CDE to CD practices, the communication of software changes through emails tend to be replaced or complemented by other practices such as the usage of shared virtual boards and tracking systems. Its usage has been described in the studies [6,19,26,47] as well by the companies B, C, D and F. One of the main advantages of email over other synchronous communication practices such as status meetings, is the fact that interested stakeholders can consume relevant information about software changes whenever they understand is necessary, what is also something Neely *et al.* [26] described in their study, where the replacement of big meetings by crisp emails in order to communicate users about new software features has been reported.

Good Practice 6 - *Using instant messaging rooms to share projects status and software changes information.*

The usage of instant messaging as a communication channel for communicating software changes has been reported in the studies [11,14,22,24,25,30,34] as well by the companies A, B, E and F. Instant Messaging practices are usually used by software development and operations teams in order to exchange additional details around software changes which could not be found in other sources. Companies such as Company A also use instant messaging channels to broadcast communication around software changes, what has been also reported by Downs *et al.* [11]. Such practice may replace the usage of emails to communicate software changes, once allow the centralisation of information in specific chat rooms.

Good Practice 7 - *Using wiki pages, blogs and collaborative portals to share software changes information.*

The practice of sharing information related to software changes through wiki pages, blogs and collaborative portals is described in the literature by the studies [8,22,26,33,34,43] as well as by the companies A, B and D. The information shared through such portals are usually available in a higher granularity of details. The advantage of this practice over other practices such as the usage of dashboards is the fact that such portals usually include collaboration features in a central space, which might be useful for several reasons. Such portals might also be integrated with Information Radiators and use information stored in change log databases in order to share relevant information regarding software changes with relevant stakeholders.

7 Conclusion

This study has assessed the challenges and impacts of ConSD practices on the communication of software changes and consolidated a set of good practices that aim to contribute to improve the communication of software changes in such environments. Initially, a SLR explored the challenges and practices related to communication of software changes between development and operations teams in ConSD environments. Additionally, an empirical study has been performed with professionals from seven global companies, which helped us to understand additional industry practices.

The overall results of this study indicate that challenges and practices related to the communication of software changes in ConSD environments vary according to a set of factors, including frequency of software changes and deploy, teams' structure and product design. As higher is the frequency of changes released, higher is the need for automatic and asynchronous communication practices to communicate software changes such as the usage of shared kanban boards and tracking systems. In the opposite side, less frequent releases allow the usage of synchronous or manual communication practices, such as structured meetings, release notes and emails. Product design is also an important factor in the process of communicating software changes. As complex is the product design, higher is the need to communicate software changes that affect its usability, including public product support documentation.

Finally, creating mechanisms to foster teams' autonomy and transparency around the communication of software changes should be a goal for those companies who want to deliver better products and services in the pace required by ConSD environments.

References

1. Aghajani, E., et al.: Software documentation issues unveiled. In: Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, pp. 1199–1210. IEEE Press (2019)
2. Alyahya, S., Ivins, W.K., Gray, W.A.: Co-ordination support for managing progress of distributed agile projects. In: IEEE Sixth International Conference on Global Software Engineering Workshop, pp. 31–34 (2011)

3. Alyahya, S., Ivins, W.K., Gray, W.A.: A holistic approach to developing a progress tracking system for distributed agile teams. In: IEEE/ACIS 11th International Conference on Computer and Information Science, pp. 503–512 (2012)
4. Ba, K., Charters, S.: Guidelines for performing systematic literature reviews in software engineering, vol. 2, January 2007
5. Brandtner, M., Giger, E., Gall, H.: Supporting continuous integration by mashing-up software quality information. In: Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), pp. 184–193 (2014)
6. Brandtner, M., Giger, E., Gall, H.: SQA-mashup: a mashup framework for continuous integration. Inf. Softw. Technol. **65**, 97–113 (2015)
7. Callanan, M., Spillane, A.: DevOps: making it easy to do the right thing. IEEE Software **33**(3), 53–59 (2016)
8. Claps, G.G., Svensson, R.B., Aurum, A.: On the journey to continuous deployment: technical and social challenges along the way. Inf. Softw. Technol. **57**, 21–31 (2015)
9. Creswell, J.: Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. SAGE Publications, New York (2009)
10. Diel, E., Marczak, S., Cruzes, D.S.: Communication challenges and strategies in distributed DevOps. In: IEEE 11th International Conference on Global Software Engineering (ICGSE), pp. 24–28 (2016)
11. Downs, J., Hosking, J., Plimmer, B.: Status communication in agile software teams: a case study. In: 2010 Fifth International Conference on Software Engineering Advances, pp. 82–87 (2010)
12. Feitelson, D.G., Frachtenberg, E., Beck, K.L.: Development and deployment at Facebook. IEEE Internet Comput. **17**(4), 8–17 (2013)
13. Gupta, R.K., Venkatachalapathy, M., Jeberla, F.K.: Challenges in adopting continuous delivery and devops in a globally distributed product team: a case study of a healthcare organization. In: ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE), pp. 30–34 (2019)
14. Itkonen, J., Udd, R., Lassenius, C., Lehtonen, T.: Perceived benefits of adopting continuous delivery practices. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016. Association for Computing Machinery, New York (2016)
15. Jurca, G., Hellmann, T.D., Maurer, F.: Agile User-Centered Design, pp. 109–123. Wiley, New York (2017). Chap. 6
16. Kim, E., Ryoo, S.: Agile adoption story from NHN. In: IEEE 36th Annual Computer Software and Applications Conference, pp. 476–481 (2012)
17. Kitchenham, B.: Procedures for performing systematic reviews, vol. 33. Keele University, Keele, UK, August 2004
18. Klepper, S., Krusche, S., Brügge, B.: Semi-automatic generation of audience-specific release notes. In: IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED), pp. 19–22 (2016)
19. Krusche, S., Alperowitz, L., Bruegge, B., Wagner, M.O.: Rugby: an agile process model based on continuous delivery. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, RCoSE 2014, pp. 42–50. Association for Computing Machinery, New York (2014)
20. Kula, E., Rastogi, A., Huijgens, H., Deursen, A.v., Gousios, G.: Releasing fast and slow: an exploratory case study at ING. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019, pp. 785–795. Association for Computing Machinery, New York (2019)

21. Lai, S., Leu, F.: Applying continuous integration for reducing web applications development risks. In: 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA), pp. 386–391 (2015)
22. Leite, L., Rocha, C., Kon, F., Milojicic, D., Meirelles, P.: A survey of DevOps concepts and challenges. ACM Comput. Surv. **52**(6) (2019)
23. Liechti, O., Pasquier, J., Reis, R.: Beyond dashboards: on the many facets of metrics and feedback in agile organizations. In: IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), pp. 16–22 (2017)
24. Luz, W.P., Pinto, G., Bonifácio, R.: Adopting DevOps in the real world: a theory, a model, and a case study. J. Syst. Softw. **157**, 110384 (2019)
25. Lwakatare, L.E., et al.: Devops in practice: a multiple case study of five companies. Inf. Softw. Technol. **114**, 217–230 (2019)
26. Neely, S., Stolt, S.: Continuous delivery? easy! just change everything (well, maybe it is not that easy). In: 2013 Agile Conference, pp. 121–128 (2013)
27. Olsson, H.H., Alahyari, H., Bosch, J.: 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, pp. 392–399 (2012)
28. Palihawadana, S., Wijeweera, C.H., Sanjitha, M.G.T.N., Liyanage, V.K., Perera, I., Meedeniya, D.A.: Tool support for traceability management of software artefacts with DevOps practices. In: Moratuwa Engineering Research Conference (MERCon), pp. 129–134 (2017)
29. Punjabi, R., Bajaj, R.: User stories to user reality: a DevOps approach for the cloud. In: IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT), pp. 658–662 (2016)
30. Rahman, A.A.U., Helms, E., Williams, L., Parnin, C.: Synthesizing continuous deployment practices used in software development. In: Proceedings of the 2015 Agile Conference, AGILE 2015, pp. 1–10. IEEE Computer Society, USA (2015)
31. Rodríguez, P., et al.: Continuous deployment of software intensive products and services: a systematic mapping study. J. Syst. Softw. **123**, 263–291 (2017)
32. Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., Stumm, M.: Continuous deployment at Facebook and OANDA. In: IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 21–30 (2016)
33. Schwarzer, J., Draheim, S., von Luck, K., Wang, Q., Casaseca, P., Grecos, C.: Ambient surfaces: interactive displays in the informative workspace of co-located scrum teams. In: Proceedings of the 9th Nordic Conference on Human-Computer Interaction, NordiCHI 2016. Association for Computing Machinery, New York (2016)
34. Senapathi, M., Buchan, J., Osman, H.: Devops capabilities, practices, and challenges: insights from a case study. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, EASE 2018, pp. 57–67. Association for Computing Machinery, New York (2018)
35. Shahin, M., Ali Babar, M., Zhu, L.: Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access **5**, 3909–3943 (2017)
36. Shahin, M., Babar, M.A., Zahedi, M., Zhu, L.: Beyond continuous delivery: an empirical investigation of continuous deployment challenges. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 111–120 (2017)

37. Shahin, M., Babar, M.A., Zhu, L.: 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, ESEM 2016. Association for Computing Machinery, New York (2016)
38. Shahin, M., Zahedi, M., Babar, M.A., Zhu, L.: An empirical study of architecting for continuous delivery and deployment. Empirical Softw. Eng. **24**(3), 1061–1108 (2019)
39. Siqueira, R., Camarinha, D., Wen, M., Meirelles, P., Kon, F.: Continuous delivery: building trust in a large-scale, complex government organization. IEEE Software **35**(2), 38–43 (2018)
40. Stettina, C.J., Heijstek, W.: Necessary and neglected? An empirical study of internal documentation in agile software development teams. In: Proceedings of the 29th ACM International Conference on Design of Communication, SIGDOC 2011, pp. 159–166. Association for Computing Machinery, New York (2011)
41. Ståhl, D., Hallén, K., Bosch, J.: Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. Empirical Softw. Eng. **22**(3), 967–995 (2017). https://doi.org/10.1007/s10664-016-9457-1
42. Ståhl, D., Hallén, K., Bosch, J.: Continuous integration and delivery traceability in industry: needs and practices. In: 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 68–72 (2016)
43. Van Heesch, U., Theunissen, T., Zimmermann, O., Zdun, U.: Software specification and documentation in continuous software development: a focus group report. In: Proceedings of the 22nd European Conference on Pattern Languages of Programs, EuroPLoP 2017. Association for Computing Machinery, New York (2017)
44. Virtanen, A., Kuusinen, K., Leppnen, M., Luoto, A., Kilamo, T., Mikkonen, T.: On continuous deployment maturity in customer projects. In: Proceedings of the Symposium on Applied Computing, SAC 2017, pp. 1205–1212. Association for Computing Machinery, New York (2017)
45. Wiedemann, A., Forsgren, N., Wiesche, M., Gewald, H., Krcmar, H.: Research for practice: the DevOps phenomenon. Commun. ACM **62**(8), 44–49 (2019)
46. Yaman, S.G.: Customer involvement in continuous deployment: a systematic literature review. In: Daneva, M., Pastor, O. (eds.) REFSQ 2016. LNCS, vol. 9619, pp. 249–265. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30282-9_18
47. Younas, M., Jawawi, D.N., Ghani, I., Fries, T., Kazmi, R.: Agile development in the cloud computing environment: a systematic review. Inf. Softw. Technol. **103**, 142–158 (2018)