Critical Success Factors of Continuous Practices in a DevOps Context

Michiel van Belzen
Open University Heerlen, The Netherlands *michiel.vanbelzen@ou.nl*

Jos Trienekens
Open University Heerlen, The Netherlands *jos.trienekens@ou.nl*

Rob Kusters
Open University Heerlen, The Netherlands *rob.kusters@ou.nl*

Abstract

Context: Software companies try to achieve adaptive near to real-time software delivery and apply continuous practices in a DevOps context. While continuous practices may create new business opportunities, continuous practices also present new challenges. Objective: This study aims to aid in adopting continuous practices and performance improvements by increasing our understanding of these practices in a DevOps context. Method: By conducting a systematic literature review we identified critical success factors on continuous practices and grouped the found factors. This led to the construction of our initial framework. We started to validate the critical success factors in this framework in a DevOps context by conducting a first pilot interview. Results: We developed an initial framework of critical success factors and conducted a pilot interview to make a first step to validate the framework. Some factors were confirmed and clarified i.e., enriched, on the basis of the retrieved information. In future work we will strive at further validation of the framework. Conclusions: We took a first step to validate our framework and retrieved valuable information, which is promising to take the next steps for further development of the framework.

Keywords: DevOps, Continuous Integration, Continuous Delivery, Continuous Deployment, Critical Success Factors.

1. Introduction

To maintain their competitive advantage, software companies need to deliver valuable product features in short cycles [5], while embracing business changes and pursuing economic efficiency [33]. In an empirical study that examines how the concept of Value is perceived in 14 agile software development organizations, the findings reveal that in general, Delivery Process with respect to time is deemed the most important value aspect among the participants [1]. Therefore the software development industry implement continuous practices that enable organizations to frequently and reliably release new features and products [10, 22, 35]. Well known continuous practices in the context of software delivery are Continuous Integration, Continuous Delivery and Continuous Deployment. Continuous Integration comprises i.a., compiling code and building software packages [12]. Continuous Delivery and Continuous Deployment enable deployments of software to production at any time [15, 33]. This implies that the software intensive industry is evolving towards a value-driven and adaptive real-time business paradigm [15, 16, 33]. For example Continuous Deployment may create new business opportunities [7]. However it also comes with new challenges e.g., technical and social challenges, and risks of adopting the Continuous Deployment process [7]. More recent studies (including literature reviews) highlight a number of challenges

which need to be overcome if continuous practices are to be successful e.g., interchangeably and synonymously used terms without rigorous definition [12, 22, 42], but also system design problems, build design and release problems, and human and organizational problems [7, 9, 22]. Based on a systematic literature review Shahin, Ali Babar and Zhu [35] extracted seven Critical Success Factors (CSFs) of continuous practices, in an order of importance: testing (effort and time), team awareness and transparency, good design principles, customer, highly skilled and motivated team, application domain, and appropriate infrastructure. However, they did not validate these factors. In addition their definition of a CSF does not conform to the CSF theory and therefore does not show much rigor. Dikert, Paasivaara and Lassenius [10] call for more research on how challenges and success factors in agile transformations are experienced and which ones are considered most important. With this study we aim to know what the rigorously validated critical success factors of continuous practices are.

In addition to continuous practices, recent studies show that DevOps adoption improve cycle times (and quality) as well [8, 27]. Although some researchers [27, 45] suggest a relation between DevOps and continuous practices, there is no well-established consensus regarding this relation [42]. With this study we aim to aid in adopting continuous practices by increasing our understanding of continuous practices in a DevOps context. This is relevant because more rigorous research on continuous practices (e.g., Continuous Deployment) is needed [33] and no consensus exists in literature on the definitions of continuous practices [12, 22].

Our main research question is: What are critical success factors of continuous practices in a DevOps context? The sub questions are: (SQ1) What are continuous practices? (SQ2) What are critical success factors of continuous practices? (SQ3) How are these factors addressed in a DevOps context? To obtain an answer to SQ1 we elaborate on the concepts of continuous practices in chapter 2. And we conducted a systematic literature review on critical success factors of continuous practices. Based on the findings we developed the initial framework of critical success factors which enabled us to give an initial answer to SQ2 (chapter 3). To validate the initial answer to SQ2 we conducted a first interview (chapter 3). This moment we can not yet answer SQ3. Therefore we point to future work that will further validate and refine the framework in chapter 4. This should lead to a full answer to SQ2 and an answer to SQ3. We hope the framework can serve as a checklist to (1) guide for achieving implementation success of continuous practices, (2) achieve performance improvements and (3) raise awareness for those involved. The framework may be used as input for a performance measurement system [17].

2. Theoretical background

Software development agility is considered as the capability of a development team to efficiently and effectively respond to and incorporate requirement changes [24]. Continuous practices extent development agility by moving from cyclic to continuous value delivery [33]. This chapter aims to answer our sub question SQ1 and to prepare on SQ2. We begin to explore theoretical perspectives and empirical findings of continuous practices. Continuous practices are expected to shorten the ‘cycle time’ i.e., minimalizing the time from the conception of a user story to its production [5]. Continuous Integration, elaborated in Section 2.1 ‘Continuous Integration’, is considered to be a predecessor and requirement of Continuous Delivery [22, 35]. We elaborate on Continuous Delivery in Section 2.2 ‘Continuous Delivery’. Some researchers [12, 22] consider Continuous Deployment as an extension to Continuous Delivery. In these cases Continuous Deployment (elaborated in Section 2.3 ‘Continuous Deployment’) is considered as a process in which each change is built, tested and deployed to production automatically in contrast to Continuous Delivery where each change requires a manual step to deploy to production [22]. However in both practices

the deployment to production is automated [5].

In Section 2.4 'Critical Success Factors', we explore theories on CSFs, which enables us to retrieve CSFs on continuous practices to answer SQ2.

2.1. Continuous Integration

Continuous Integration may be defined as a process which comprises inter-connected steps such as compiling code, running unit and acceptance tests, validating code coverage, checking compliance with coding standards, and building deployment packages [12]. The process is typically automatically triggered and is further characterized by frequency of integration, ceremonies and highly visible artifacts to help ensure that problems leading to integration failures are solved as quickly as possible [12], and automated integration flows [39]. The frequency of integration should be regular enough to ensure quick feedback to developers [12]. Good practice of Continuous Integration requires all developers to integrate their work to a common code repository, usually at least daily [13, 22]. After each integration, the system should be built and tested, to ensure that the system is still functional after each change and that it is safe for others to build on top of the new changes [22]. This iteration of integrating, testing source code and solving problems is a 'feedback loop'. In this way the level of confidence in the source changes may successively increase as they progress through the system [39]. However Continuous Integration activities are carried out much more infrequently in industry practice as the term 'Continuous Integration' suggests [39], especially at an individual level [43]. Larger software size, larger organizational size, a lower proportion of developers in the organization and bigger software modules correlate with lower continuity [15]. In addition Ståhl, Mårtensson and Bosch [43] suggest that from a continuity point of view, direct integration with the mainline is superior, but that larger organizations are unable or unwilling to work in such a way. In 2013 Ståhl & Bosch found differences in experienced Continuous Integration effects [40]. In addition they concluded in 2014 based on a literature review, that there was no consensus on Continuous Integration as a single, homogeneous practice [41]. Therefore the authors proposed a descriptive model for better documentation of Continuous Integration variants. Their model contains a number of attributes, grouped into themes, covering the variation points where Continuous Integration implementations differ.

2.2. Continuous Delivery

Continuous Delivery is considered a software engineering approach [5]. It is composed of a set of principles, patterns and practices designed to make deployments at any time [14]. Teams continuously produce valuable software in short cycles, keeping the software in a certain state till the point a human decides to release, to be able to release reliably at any time [5, 15, 22]. Or in other words, as Fitzgerald and Stol [12] noticed, Continuous Delivery refers to the ability to deploy software to an environment. Important characteristics of Continuous Delivery are: valuable software, short cycles, releasable at any time, and reliable releases [5]. Other important aspects are fast feedback loops, work in small batches and automation support to ensure quick and reliable releases to customers [12, 14, 35]. The aim of Continuous Delivery is reducing risks and transaction costs while producing valuable software to production [14].

Laukkanen, Itkonen and Lassenius [22] found problems, causes and solutions when adopting Continuous Delivery. The researchers conclude that problems of system design are common, critical and the largest problems. In addition they noticed that system design problems as well as resource and human and organizational problems have the most effect on other themes. This is acknowledged by Humble [14] who states that the real obstacles to implementing Continuous Delivery are the inadequate architecture and a nongenerative culture. To help overcome adoption challenges Chen [5] presented six strategies: (1) selling Continuous Delivery as a painkiller; (2) establishing a dedicated team with multi-disciplinary members; (3) Continuous

Delivery of Continuous Delivery; (4) starting with the easy but important applications; (5) visual Continuous Delivery pipeline skeleton; (6) expert drop.

2.3. Continuous Deployment

In contrast to Continuous Delivery, the Continuous Deployment approach aims to deploy software code immediately to production for customers to use [7, 15, 33]. No manual steps or decisions between a developer commit and a production deployment are needed [22]. In this way feedback from users is gained much faster, reducing costs of detecting defects [14, 22]. According to Humble [14] Continuous Deployment is mainly limited to cloud- or datacenter- hosted services.

Besides customer feedback, Continuous Deployment requires (1) agile processes at the team level; (2) integration of the complete R&D organization; (3) parallelization and automation of processes [33].

2.4. Critical Success Factors

Rockart (1979), one of the researchers who developed the CSF theory, defined a CSF as the limited number of areas in which satisfying results will ensure successful competitive performance for the organization [32]. Considering several definitions, Leidecker and Bruno [25] defined a CSF as those characteristics, conditions, or variables that when properly sustained, maintained, or managed can have a significant impact on the success of a firm competing in a particular industry. Shahin, Ali Babar and Zhu [35] classified a factor as critical for making continuous practices successful, when the factor was cited in at least 20% of the reviewed studies. However according to Ram, Corkindale and Wu [31] a factor can only be termed a CSF if attending to this factor in a satisfactory manner results in performance improvements i.e., identifying a possibly important factor is not sufficient to constitute a CSF. The authors noticed that to establish whether a CSF is really critical depends on the extent in which success and performance may be measured e.g., user satisfaction or successful deployment of software [31]. Following Ram, Corkindale and Wu [31] the question is whether the factors that have been identified in the literature as CSFs of continuous practices have been well enough established empirically as contributing to implementation success and/or performance outcome. This is important as decision making by management becomes complex when many factors are to be considered to achieve the desired goal of continuous value delivery [18]. In this study we define a CSF as a factor which: (1) should have a verified significant impact on the success of a continuous practice; (2) results in verified performance improvements; (3) success and performance may be measured.

Now that we have elaborated on continuous practices and CSF theories, we are able to answer sub question SQ1 ‘What are continuous practices?’. Based on state of the art literature we consider Continuous Integration as an automatically triggered process which automatically integrates highly visible artifacts frequently enough to ensure quick feedback to developers. Similarly we consider Continuous Delivery as the ability, supported by automation, to reliably deploy valuable software to a production environment at any time when a human decides to do so with the aim to reduce risks and transaction costs. Continuous Deployment is defined almost the same, except the actual deployment is done automatically instead of triggered by a human.

We explored CSF theories to prepare on SQ2 ‘What are critical success factors of continuous practices’. Based on literature we found several criteria which we used to formulate our definition of a CSF as a factor which: (1) should have a verified significant impact on the success of a continuous practice; (2) results in verified performance improvements; (3) success and performance may be measured.

3. Towards a Framework of Success Factors of Continuous Practices

To meet the call for more rigorously conducted research [33], we commenced a Systematic Literature Review (SLR) method which is a methodologically rigorous review of research results [19]. We choose to conduct a conventional SLR as a means to aggregate knowledge about our research question [20], because we have a fine grained research question (we search for CSFs). We structured the results of the SLR into a preliminary theoretical framework of CSFs. We conducted pilot interviews with experts in the field of continuous practices active in DevOps teams to validate the framework.

3.1. Research Methodology

Search process

Following the search process according to Kitchenham et al. (2010) [20] we searched six digital libraries and one broad indexing service: IEEE computer society digital library, ACM digital library, SpringerLink, Web of Science, Citeseer and SCOPUS. Searches were based on 'All fields' (ACM, SpringerLink and Web of Science), 'abstract' (IEEE, Citeseer), 'Title, abstract and keywords' (SCOPUS) and the period between 2001 (the emerge of the agile manifesto) and June 2019. We used the following search string for every library (in the case of Citeseer we used three separate search strings: "Continuous Delivery" AND DevOps, "Continuous Deployment" AND DevOps, "Continuous Integration" AND DevOps) and indexing service: ("Continuous Delivery" OR "Continuous Deployment" OR "Continuous Integration") AND DevOps. We did not include "Critical Success Factor" or "Factor" to the search string, because we expected to find fewer relevant papers. The number of papers found, was 36 (IEEE), 213 (ACM), 113 (SpringerLink), 17 (Web of Science), 369 (Citeseer) and 177 (SCOPUS). We validated the outcome against the papers found during our exploratory research and found one additional and relevant paper: Laukkanen et al. (2017) [22].

Study selection & quality assessment

After integrating the results for the different searches (resulting in 909 found papers) and the removal of duplicates (resulting in 825 papers), we undertook an initial screening of the remaining papers based on abstract including papers which studied continuous practices topics (resulting in 82 papers). During the screening we exclude studies that were obviously irrelevant and not in the English language. Afterwards we assessed the remaining papers by reading the full text on the basis that they did not include success/risk/fail factors. After reading the papers, 36 papers did not contain CSFs leaving 19 papers for data extraction.

Data extraction process

In addition to the elaborated quality assessment above, the following data was extracted from the remaining papers: theme, name of the factor (i.e., problem or challenge or action or success factor or barrier or obstacle), citation of the factors' description and paper reference. We removed duplicates during the extraction.

3.2. Research Results

We summarized the description (citation) of every factor and grouped the found factors into themes adopted from Laukkanen et al. [22] and Shahin, Ali Babar and Zhu [35], which resulted in Table 1 'Initial framework'. Some factors could be related to specific themes similar to Laukkanen et al. [22] and Shahin, Ali Babar and Zhu [35], viz. build design (factors that were caused by build design decisions), system design (factors that were caused by system design decisions), integration (factors that arise when the source code is integrated into the mainline), testing (factors related to software testing), release (factors occurring when the software is released), resource (factors related to resources), customer (factors that arise from customer requirements or circumstances). Other factors are generic or correspond with organizational aspects. We related these factors to the theme 'human and organizational', similar to Laukkanen et al. [22] and Shahin, Ali Babar and Zhu [35] as well.

Table 1 Initial framework

Factor	Description	Reference
Theme: Build design (factors that were caused by build design decisions).		
Complex build	Build system, process or scripts are complicated or complex.	[22]
Inflexible build	The build system cannot be modified flexibly.	[22]
Legacy code considerations	Integration with legacy systems sometimes enforce traditional test processes and/or strain the legacy system developers (e.g. legacy systems' architectures are usually not amenable to Continuous Delivery and the teams working on these legacy systems usually have 'legacy' culture, practices, and mindsets).	[2, 5, 26]
Application architecture	(re-) architect applications for Continuous Delivery. Software components or services, but also every entity (e.g., database and operating system) that the application depends on, should be a unit of deployment in the deployment process.	[5, 6, 37]
Fast feedback	Developers are not able to receive the feedback from tests quickly.	[12, 35]
Theme: System design (factors that were caused by system design decisions).		
Database schema changes	Software changes require changes of database schema.	[22, 35]
Dependencies in design and code	Highly coupled architectures, difficulty to find autonomous requirements for frequent integrations.	[35]
Internal dependencies	Dependencies between parts of the software system.	[22]
System modularization	The system consists of multiple units e.g., modules or services.	[22]
Unsuitable architecture	System architecture limits Continuous Delivery.	[22]
Vendor lock-in	A situation in which a customer using a product or service cannot easily transition to a competitor's product or service.	[4]
Theme: Integration (factors that arise when the source code is integrated into the mainline).		
Broken build	Build stays broken for long time or breaks often.	[22]
Broken development flow	Developers get distracted and the flow of development breaks.	[22]
Customer environment	Lack of access to customer environment, complex and manual configuration, diversity and complexity of customer sites, difficulty to stimulate production-like environment.	[35]
Dependency at application level	Organizations need to ensure that there is no integration problem when deploying an application to production.	[35]
Different development and production environments	Different environments cause frustrations due to non-representative testing.	[26]
Domain constraints	Some domains do not allow or cause difficulties to truly adopt and implement Continuous Deployment.	[26, 35, 36]
Internal verification loop	Needs to be shortened in order to not only develop functionality fast but also deploy it fast at customer site.	[29]
Large commits	Commits containing large amount of changes.	[22]
Long-running branches	Code is developed in branches that last for long time.	[22]
Merging conflicts	Third party components, incompatibly among dependent components, lack of understanding about changed components.	[22] [35]
Network configuration and upgrade complexity	Different network configurations at customer site cause upgrade complexity.	[29]
Slow integration approval	Changes are approved slowly to the mainline.	[22]
Work blockage	Completing work tasks is blocked or prevented by broken build or other integrations in a queue.	[22]
Theme: Testing (factors related to software testing).		
Ambiguous test result	Test result is not communicated to developers, is not an explicit pass or fail or it is not clear what broke the build.	[22]
Complex testing	Testing is complex e.g., setting up environment.	[22]
Flaky tests	Tests that randomly fail sometimes.	[22]
Hardware testing	Testing with special hardware that is under development or not always available.	[22]
Insufficient level of automated test coverage	Lack of sufficient automated test coverage	[36]
Lack of fully automated user acceptance test	Automation of tests at the end of the development process e.g., (user) acceptance test and performance test, requires heavy workloads and time.	[36]
Lack of proper test strategy	Lack of fully automated testing, lack of test-driven development.	[35]
Manual and nonfunctional testing	Additional test procedures required to cover specific tests and nonfunctional tests, hindering test automation.	[5, 22]
Manual interpretation of test results	Depends on the extent the regression tests can be automated, organizations can significantly reduce the overall cycle time.	[36]
Manual quality check	Although automation is critical in Continuous Deployment practice, manual tasks are	[36]

Factor	Description	Reference
	sometimes unavoidable.	
Multi-platform testing	Testing with multiple platforms when developers do not have access to all of them.	[22]
Poor test quality	Instable tests, low test coverage, low quality test data, long running tests, test dependencies.	[35]
Problematic deployment	Deployment of the software is time-consuming or error-prone.	[22]
Time-consuming testing	Testing takes too much time. The significantly increased frequency of test execution makes test optimization important and useful.	[5, 22, 28]
UI testing	Testing the user interface of the application.	[22]
Untestable code	Software is in a state that it cannot be tested.	[22]
Theme: Release (factors occurring when the software is released).		
Customer data preservation	Preserving customer data between upgrades.	[22]
Deployment downtime	Downtime cannot be tolerated with frequent releases.	[22]
Documentation	Keeping the documentation in-sync with the released version.	[22, 38]
Feature discovery	Users might not discover new features.	[22]
Highly bureaucratic deployment process	Process which has a large number of formal tasks (e.g., getting approvals from various people) to be performed manually before each release.	[36]
Lack of efficient roll back mechanism	Lack of having efficient rollback mechanism, forces an organization to decrease the pace of pushing changes to production.	[36]
Marketing	Marketing versionless system.	[22]
More deployed bugs	Frequent releases cause more deployed bugs.	[22]
Third party integration	Frequent releases complicate third party integration.	[22]
Theme: Human and organizational (factors related to human and organizational aspects).		
Changing roles	Different roles need to adapt for collaboration.	[22]
Coordination and collaboration challenges	Practicing Continuous Integration, Continuous Delivery, Continuous Deployment needs more and effective coordination and communication between team members.	[35]
Cost	Major upgrade in infrastructure and resources, training and coaching.	[35]
Difficulty to change established organizational policies and cultures	Lack of agile and suitable business model, changing log-lived feature branching to short-live one in an established company.	[35]
Distributed organization	Distributed team model, inconsistent perceptions among team members.	[35]
Increase of technical debt	Omitted quality and shortcuts in the development process.	[47]
Lack of awareness and transparency	Lack of understanding about the status of a project increase the number of merge conflicts.	[35]
Lack of discipline	Discipline to commit often, test diligently, monitor the build status and fix problems as a team.	[22]
Lack of experience	Lack of experience practicing Continuous Integration or Continuous Deployment. Misunderstanding and trying to solve the problem with a prescriptive methodology or a big expensive all-in-one DevOps technology. Or considering moving faster as compromising the quality.	[22, 30, 35]
Lack of motivation	People need to be motivated to get past early difficulties and effort.	[22]
Lack of transparency	The need to get an overview of the current status of development projects.	[29]
More pressure	Increased pressure because software needs to be in always-releasable state.	[22, 35]
Organizational structure	E.g., separation between divisions causes problems. Siloed lines of business.	[22, 30]
Own interests	Tension exists between departments due to competing goals.	[4]
Own way of working	Process sub-optimization	[4]
Perceived territories of control	E.g., system access required for Continuous Delivery is controlled by other parts of the organization that do not have a principal interest in Continuous Delivery. Sense of responsibility, which highlights that developers do not show accountability for the deployment of the software or product.	[4, 30]
Resistance to change	Not being facilitated in the change, new way of working not fitting the culture, unexpected social and technical implications.	[22, 30, 35]
Skepticism and distrust on continuous practices	Lack of trust on benefits of Continuous Integration, Continuous Delivery, Continuous Deployment.	[35]
Team coordination	Increased need for team coordination.	[22, 35]
Team dependencies	Cross-team dependencies, ripple effects of changes on multiple teams, dependency between feature team and module team in embedded system domain.	[35]
Waste in processes	Many traditional processes hinder Continuous Deployment e.g., a feature ready for release must go through a change advisory board.	[4, 11]
Theme: Resource (factors related to resources).		
Developer trust and confidence	Developers must have sufficient proficiency and knowledge of typical Continuous Deployment.	[26]
Effort	Initially setting up Continuous Delivery requires effort.	[22]
Insufficient hardware	Build and test environments require hardware resources.	[7, 22]

Factor	Description	Reference
resources		
Infrastructure support	The way the infrastructure is managed. Lack of the capability to automatically provision the required infrastructure. The use of Platform-as-a-Service (PaaS).	[5, 21]
Lack of suitable tools and technologies	Lack of mature tools for automating tests and reviewing code in Continuous Integration, frequent changes in tools, security and reliability issues in build and deployment tools, current tools do not fit to all organizations. Automated DevOps pipeline security.	[3, 5, 30, 35]
Network latencies	Network latencies hinder Continuous Integration.	[22]
Theme: Customer (factors that arise from customer requirements or circumstances).		
Applications that are not amenable to Continuous Deployment	For example monolithic, large applications.	[4]
Communication	Communication with the customer e.g., establishing trust before collaboration, choosing the right form of communication strategy, dealing with conflicts and with different stakeholders.	[46]
Customer environment (dealing with regulations)	Lack of carefully studying and exploring customer environments before moving to Continuous Deployment leads to challenges e.g., dealing with regulations, or enforced network separation.	[23, 36]
Customer preference	Not all customers are happy with frequent release, customer organization policy may affect practicing Continuous Deployment.	[26, 35]
Customer profile	The needs of different user groups might diverge and change in different contexts, establishing a customer sample group where all possible types of users are represented, the customer's level of competence, experience, knowledge and/or reliability.	[46]
Data management	The customer-related data collection process and analysis of the data e.g., the internal verification loop of the collected data has to be short and systematic, feedback should be coming from the right channel. The analysis process requires high effort e.g., to work with data with noise in it, eliminate human factors such as subjectivity or prioritize tasks.	[46]
Demotivated customer	Time and pace of deployment to production greatly depends customers' cultures, polices and goals. Not all customers are mature enough to accept a continuous release.	[36, 46]
Dependencies with hardware and other (legacy) application	Releasing an application on continuous basis requires deploying all dependent applications in customer site, hardware and network dependencies.	[35]
Deployment considered as a business decision	Therefore development team members have little control over deployment to production.	[36]
Experiments and A/B testing	The customer might not want to be a part of an experiment or they might not welcome partially developed functionality. Determining where to start to experiment with the customer.	[46]
Sales and suppliers	Intermediaries might not be interested in collecting customer feedback after selling a product or a service. Therefore user data is not accessible.	[46]
Setting-the-scene	Preparing and receiving customer input is time-consuming.	[46]
Transparency	Limited or no transparency in data, process and feedback demotivates users to provide feedback. However, too much transparency causes customers to interfere with developers' work, failures might be too visible to customers.	[46]
Updates, new features and products	Customers might not realize or welcome changes.	[22, 46]

We chose an inductive approach i.e., carrying out (cross-sectional) semi-structured interviews and document studies to validate the factors of Table 1 'Initial framework'. This approach contributes to our research goal to increase our understanding of continuous practices in a DevOps context, because it enables us to achieve depth, elaboration and soundness [44]. In addition it gives us the possibility to address real and concrete experiences. We will interview experienced (preferably at least five years of experience) DevOps team members and corresponding managers, because our initial framework contains factors on different themes (technical and organizational). The interviews will be transcribed, shared with the participants for feedback and finally used for data analysis. To guide the interviews, we developed an interview protocol. The interview protocol contains the steps to take from the invitation up to and including the interview and required documents, viz. invitation letter, letter of consent and questionnaire. The definitions of our concepts, Table 1 'Initial framework', and the letter of consent are part of the invitation letter. Thus enabling the participants to prepare themselves. To test our interview protocol we conducted a pilot interview. Thus we selected an experienced DevOps team lead, who was willing to participate. We sent

the team lead the invitation letter, including the definitions and the initial framework. The initial framework was used as guidance/structure during the interview, which worked out very well. We asked the interviewee if a success factor was recognized and if it could be supported by a real experience. We also asked how the interviewee estimates the importance/weigh of the factor, if there are any additional success factors and which factors are related to each other (and why). We reserved 90 minutes to interview the team lead and discussed all the factors. Some factors took more time than others. However we did find some interesting results. The factors concerning the themes of build design, system design and integration were considered as CSFs by the interviewee, because a low code development platform was used, which enforces correct builds contributing significant to Continuous Integration. Factors belonging to the theme of human and organization were not considered as critical, with the exception of technical debt. Lack of awareness by the product owner on technical debt impede continuous practices in the long term. Concerning the theme of testing, the level of automation was considered to be low impeding continuous practices. However some tests were expected to continue to be manual e.g., a user acceptance test. Documentation was seen as an important CSF as well, because undocumented features confuse both developers and users. On the theme of release, third party integration was impeding continuous practices in the past (at the moment a third party is not needed anymore) and therefore mentioned as a CSF. And on the theme of customer, communication was mentioned as a CSF due to the lack of a communication process, which should support communication between the customer and developers. The interviewee mentioned the role/responsibility of the product owner several times, emphasizing that this is an important role. This could affect the factors 'Customer preference' and 'Customer profile', which are part of the theme 'Customer'. It would appear that in this case these factors weigh more than other discussed factors.

During the interview we realized that some factors appear to be the same e.g., coordination and collaboration challenges, distributed organization, organizational structure, perceived territories of control, team coordination, and team dependencies. These factors are related to the way an organization is structured. Finally the interviewee mentioned no additional factors.

To summarize our findings we found an extensive list of factors which we classified into eight themes adopted from Laukkanen [22] and Shahin, Ali Babar and Zhu [35], which resulted in our initial framework. Subsequently we conducted a pilot interview to test our interview protocol and to validate our initial framework. We found that the discussed factors were recognized and understood by the interviewee. We learned that some factors were considered as CSF, some appear to be double and some seem to weigh more than others.

4. Conclusions and Future Work

We conducted an SLR to find CSFs of continuous practices and developed an initial framework. Subsequently we conducted a pilot interview and found that the discussed factors were recognized and understood by the interviewee i.e., some factors were considered as critical. And we learned that several factors appear to be double and some factors seem to weigh more than others. Considering we extracted the found factors from studies which have different abstraction levels, duplicates appear to be evident. Besides that we decided to take more time to discuss the factors more extensive. In addition we have to expand the initial answer to SQ2 'What are critical success factors of continuous practices?' and obtain an answer to SQ3 'How are these factors addressed in a DevOps context?'. Therefore future work will be: conducting more interviews, improving our classification of the found factors using the metaplan-method [34], reformulating the remaining factors in terms of success factors to address the call for more rigorous research [33] and the operationalisation of the framework. We hope the

framework can serve as a checklist to guide for achieving implementation success of continuous practices, to achieve performance improvements and raise awareness for those involved. A performance measurement system may be used to demonstrate that an organization is following the CSFs, to monitor progress and to drive improvement by developing adequate measures [17].

We will conduct interviews in different organizations to prove the applicability of the framework. Additional empirically validated factors will be added and factors which do not have significant impact on the success of a continuous practice and which do not result in performance improvements (according our definition of a CSF), will be removed from our initial framework. This will lead to an improved and refined theoretical framework.

References

1. Alahyari, H., Berntsson Svensson, R., Gorschek, T.: A study of value in agile software development organizations. Journal of Systems and Software. 125 271–288 (2017)
2. Albuquerque, A.B., Cruz, V.L.: Implementing DevOps in Legacy Systems. In: Silhavy, R., Silhavy, P., and Prokopova, Z. (eds.) Intelligent Systems in Cybernetics and Automation Control Theory. pp. 143–161. Springer International Publishing, Cham (2019)
3. Bass, L., Holz, R., Rimba, P., Tran, A.B., Zhu, L.: Securing a Deployment Pipeline. In: 2015 IEEE/ACM 3rd International Workshop on Release Engineering. pp. 4–7. IEEE, Florence, Italy (2015)
4. Chen, L.: Continuous Delivery: Huge Benefits, but Challenges Too. IEEE Software. 32 (2), 50–54 (2015)
5. Chen, L.: Continuous Delivery: Overcoming adoption challenges. Journal of Systems and Software. 128 72–86 (2017)
6. Chen, L.: Towards Architecting for Continuous Delivery. In: 2015 12th Working IEEE/IFIP Conference on Software Architecture. pp. 131–134. IEEE, Montreal, QC, Canada (2015)
7. Claps, G.G., Berntsson Svensson, R., Aurum, A.: On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology. 57 21–31 (2015)
8. Colomo-Palacios, R., Fernandes, E., Soto-Acosta, P., Larrucea, X.: A case analysis of enabling continuous software deployment through knowledge management. International Journal of Information Management. 40 186–189 (2018)
9. Debbiche, A., Dienér, M., Berntsson Svensson, R.: Challenges When Adopting Continuous Integration: A Case Study. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., and Raatikainen, M. (eds.) Product-Focused Software Process Improvement. pp. 17–32. Springer International Publishing, Cham (2014)
10. Dikert, K., Paasivaara, M., Lassenius, C.: Challenges and success factors for large-scale agile transformations: A systematic literature review. Journal of Systems and Software. 119 87–108 (2016)
11. Farid, A.B., Helmy, Y.M., Bahloul, M.M.: Enhancing Lean Software Development by using Devops Practices. ijacsa. 8 (7), (2017)
12. Fitzgerald, B., Stol, K.-J.: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software. 123 176–189 (2017)
13. Fowler, M., Foemmel, M.: Continuous integration, http://www. thoughtworks. com/Continuous Integration. pdf, 122, 14., (2006)
14. Humble, J.: Continuous delivery sounds great, but will it work here? Communications of the ACM. 61 (4), 34–39 (2018)
15. Humble, J., Farley, D.: Continuous delivery: reliable software releases through build, test, and deployment automation. Addison-Wesley, Upper Saddle River, NJ (2010)
16. Järvinen, J., Huomo, T., Mikkonen, T., Tyrväinen, P.: From Agile Software Development to Mercury Business. In: Lassenius, C. and Smolander, K. (eds.) Software Business. Towards Continuous Value Delivery. pp. 58–71. Springer International Publishing, Cham (2014)
17. Kanji, G.K.: Performance measurement system. Total Quality Management. 13 (5), 715–728 (2002)
18. Kannan, D.: Role of multiple stakeholders and the critical success factor theory for the sustainable supplier selection process. International Journal of Production Economics. 195

391–418 (2018)
19. Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering – A systematic literature review. Information and Software Technology. 51 (1), 7–15 (2009)
20. Kitchenham, B., Pretorius, R., Budgen, D., Pearl Brereton, O., Turner, M., Niazi, M., Linkman, S.: Systematic literature reviews in software engineering – A tertiary study. Information and Software Technology. 52 (8), 792–805 (2010)
21. Krancher, O., Luther, P., Jost, M.: Key Affordances of Platform-as-a-Service: Self-Organization and Continuous Feedback. Journal of Management Information Systems. 35 (3), 776–812 (2018)
22. Laukkanen, E., Itkonen, J., Lassenius, C.: Problems, causes and solutions when adopting continuous delivery—A systematic literature review. Information and Software Technology. 82 55–79 (2017)
23. Laukkarinen, T., Kuusinen, K., Mikkonen, T.: DevOps in Regulated Software Development: Case Medical Devices. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER). pp. 15–18. IEEE, Buenos Aires, Argentina (2017)
24. Lee, G., Xia, W.: Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data. MIS Q. 34 (1), 87–114 (2010)
25. Leidecker, J.K., Bruno, A.V.: Identifying and using critical success factors. Long Range Planning. 17 (1), 23–32 (1984)
26. Leppanen, M., Makinen, S., Pagels, M., Eloranta, V.-P., Itkonen, J., Mantyla, M.V., Mannisto, T.: The highways and country roads to continuous deployment. IEEE Software. 32 (2), 64–72 (2015)
27. Lwakatare, L.E., Kuvaja, P., Oivo, M.: Relationship of DevOps to Agile, Lean and Continuous Deployment. In: Abrahamsson, P., Jedlitschka, A., Nguyen Duc, A., Felderer, M., Amasaki, S., and Mikkonen, T. (eds.) Product-Focused Software Process Improvement. pp. 399–415. Springer International Publishing, Cham (2016)
28. Marijan, D., Sen, S.: DevOps Enhancement with Continuous Test Optimization. In: The 30th International Conference on Software Engineering and Knowledge Engineering. pp. 536–585. (2018)
29. 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: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications. pp. 392–399. IEEE, Cesme, Izmir, Turkey (2012)
30. Qumer Gill, A., Loumish, A., Riyat, I., Han, S.: DevOps for information management systems. VINE Journal of Information and Knowledge Management Systems. 48 (1), 122–139 (2018)
31. Ram, J., Corkindale, D., Wu, M.-L.: Implementation critical success factors (CSFs) for ERP: Do they contribute to implementation success and post-implementation performance? International Journal of Production Economics. 144 (1), 157–174 (2013)
32. Rockart, J.F.: Chief executives define their own data needs. Harvard business review. 57 (2), 81–93 (1979)
33. Rodríguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M.: Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software. 123 263–291 (2017)
34. Schnelle, E.: The Metaplan-Method communication tools for planning and learning groups. Metaplan-GmbH, Quickborn (1979)
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: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 111–120. IEEE, Toronto, ON (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 '16.

pp. 1–10. ACM Press, Ciudad Real, Spain (2016)
38. Souza, R., Oliveira, A.: GuideAutomator: Continuous Delivery of End User Documentation. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER). pp. 31–34. IEEE, Buenos Aires (2017)
39. Ståhl, D., Bosch, J.: Automated software integration flows in industry: a multiple-case study. In: Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014. pp. 54–63. ACM Press, Hyderabad, India (2014)
40. Ståhl, D., Bosch, J.: Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study. In: Artificial Intelligence and Applications / 794: Modelling, Identification and Control / 795: Parallel and Distributed Computing and Networks / 796: Software Engineering / 792: Web-based Education. ACTAPRESS, Innsbruck, Austria (2013)
41. Ståhl, D., Bosch, J.: Modeling continuous integration practice differences in industry software development. Journal of Systems and Software. 87 48–59 (2014)
42. Stahl, D., Martensson, T., Bosch, J.: Continuous practices and devops: beyond the buzz, what does it all mean? In: 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 440–448. IEEE, Vienna (2017)
43. Ståhl, D., Mårtensson, T., Bosch, J.: The continuity of continuous integration: Correlations and consequences. Journal of Systems and Software. 127 150–167 (2017)
44. Verschuren, P., Doorewaard, H.: Designing a Research Project. Eleven International Publishing (2010)
45. Wettinger, J., Breitenbücher, U., Falkenthal, M., Leymann, F.: Collaborative gathering and continuous delivery of DevOps solutions through repositories. Computer Science - Research and Development. 32 (3–4), 281–290 (2017)
46. Yaman, S.G., Sauvola, T., Riungu-Kalliosaari, L., Hokkanen, L., Kuvaja, P., Oivo, M., Männistö, T.: Customer Involvement in Continuous Deployment: A Systematic Literature Review. In: Daneva, M. and Pastor, O. (eds.) Requirements Engineering: Foundation for Software Quality. pp. 249–265. Springer International Publishing, Cham (2016)
47. Yli-Huumo, J., Maglyas, A., Smolander, K.: The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., and Raatikainen, M. (eds.) Product-Focused Software Process Improvement. pp. 93–107. Springer International Publishing, Cham (2014)