Empirical Software Engineering (2022) 27: 157
https://doi.org/10.1007/s10664-022-10191-w

Towards cost-benefit evaluation for continuous software engineering activities

Eriks Klotins[1] · **Tony Gorschek**[1,2] · **Katarina Sundelin**[3] · **Erik Falk**[4]

Accepted: 17 May 2022 / Published online: 16 August 2022

Abstract

Context: Software companies must become better at delivering software to remain relevant in the market. Continuous integration and delivery practices promise to streamline software deliveries to end-users by implementing an automated software development and delivery pipeline. However, implementing or retrofitting an organization with such a pipeline is a substantial investment, while the reporting on benefits and their relevance in specific contexts/domains are vague.

Aim: In this study, we explore continuous software engineering practices from an investment-benefit perspective. We identify what benefits can be attained by adopting continuous practices, what the associated investments and risks are, and analyze what parameters determine their relevance.

Method: We perform a multiple case study to understand state-of-practice, organizational aims, and challenges in adopting continuous software engineering practices. We compare state-of-practice with state-of-the-art to validate the best practices and identify relevant gaps for further investigation.

Results: We found that companies start the CI/CD adoption by automating and streamlining the internal development process with clear and immediate benefits. However, upgrading customers to continuous deliveries is a major obstacle due to existing agreements and customer push-back. Renegotiating existing agreements comes with a risk of losing customers and disrupting the whole organization.

Conclusions: We conclude that the benefits of CI/CD are overstated in literature without considering the contextual and domain complexities rendering some benefits infeasible. We identify the need to understand the customer and organizational perspectives further and understand the contextual requirements towards the CI/CD.

Communicated by: Massimiliano Di Penta

✉ Eriks Klotins
eriks.klotins@bth.se

[1] Software Engineering Research Lab, Blekinge Institute of Technology, Karlskrona, Sweden

[2] fortiss, Munich, Germany

[3] Ericsson AB Karlskrona, Karlskrona, Sweden

[4] Telia Company Uppsala, Uppsala, Sweden

Keywords Continuous Integration · Continuous Delivery · Continuous Software Engineering · Cost-benefit analysis

1 Introduction

Many software companies aim to benefit from continuous software integration and delivery (CI/CD) practices. In a nutshell, CI/CD practices promise to increase the speed of delivering new features to end-users by automating steps of software delivery, such as testing, integration, and deployment to the business environment (Fitzgerald and Stol 2017; Kumar and Mishra 2016). The idea is that new features and changes are immediately and automatically delivered to end-users without shelving them and waiting for a scheduled release. Rapid delivery of incremental features has the potential to enable fast feedback cycles, continuous experimentation, and to close the gap between software developers and users (Fagerholm et al. 2017; Zhu et al. 2016).

Retrofitting an organization with A CI/CD pipeline is a substantial undertaking. Most existing software engineering practices and architectures need to be adjusted to fit within a continuous pipeline. Other organizational aspects, such as the business model and customer agreements, need to be tailored to support and benefit from continuous software delivery (Fitzgerald and Stol 2014; Neely and Stolt 2013; Chen 2015).

Organizational changes, the introduction of new ways of working and delivering value may disrupt the organization (Giardino et al. 2015). Maintenance of the pipeline, such as tooling, test data, and automated test suites, requires commitment and continuous investments (Sundelin et al. 2020; Garousi and Felderer 2016). However, the risk and investments associated with the adoption of CI/CD are justified only if the benefits are relevant and sufficiently contribute to achieving organizational objectives.

There is a substantial body of knowledge around how to implement elements of a CI/CD pipeline and overcome the associated challenges; see, for example, Olsson et al. (Olsson et al. 2012), Rodriguez et al. (Rodríguez et al. 2017), Fitzgerald et al. (Fitzgerald and Stol 2017) and Laukkanen et al. (Laukkanen et al. 2017). However, very few studies address the holistic cost/benefit aspect of CI/CD. The understanding of costs and benefits is required to make informed decisions of what parts of CI/CD are relevant for a given product and organization. Especially, if the investments are substantial and changes carry organizational risks.

Naturally, organizations aim to maximize the benefits of their investments. However, we observe that the adoption of CI/CD practices is often driven by hype and promises of huge, however vague, benefits. For example, CI/CD are ought to enable more frequent software releases and flexible business models (Humble and Kim 2018). However, how to assess the relevance of such benefits for a given company, and gauge whether the investments can be justified with the expected gains is not known. Without a detailed analysis of required investments, the relevance of the benefits, and any downsides, organizations may waste resources on implementing irrelevant practices, too expensive to maintain or even harmful to a successful business.

Organizations may also miss capitalizing on the benefits of CI/CD while incurring the cost. This is due to the potentially asymmetrical nature of costs and benefits. The costs could be incurred at one stage of the software delivery pipeline with a small perceived immediate benefit. However, the investment's actual benefits could be reaped at a later stage, only if such a later stage is implemented and aligned correctly (Laukkanen et al. 2015). The same is true upstream. For example, achieving regular and continuous customer feedback due to the pipeline might significantly benefit product management and product planning. However, if that part of the organization is not prepared to utilize said feedback as a part of their work, significant parts of the benefits are not realized.

The term CI/CD largely refers to automation aspects of software engineering. However, Fitzgerald et al. (Fitzgerald and Stol 2017) points out that there are other continuous activities, such as planning, support from the organization, business models and customers, that enable and maximize benefits of the automation efforts. Thus, we explore continuous software engineering from a holistic cost-benefit perspective encompassing all connected aspects. We map state-of-practice in two companies to state-of-the-art to identify aims, expected benefits, investments and prerequisites to adopting specific CI/CD practices. We further discuss the relevance of the potential benefits to our studied cases and highlight areas with insufficient support from state-of-the-art.

The rest of this paper is structured as follows. Section 2 presents background and related work, Section 3 outlines the research methodology, Sections 4-5 presents and interprets our findings, Section 6 concludes the paper.

2 Background and Related Work

2.1 Agile, Lean and Continuous Software Engineering

Agile software engineering principles emphasize working software over extensive planning and documentation, rapid response to change, and collaboration versus contract negotiation (Fowler et al. 2001). These principles have inspired a large number of agile software engineering practices (Jalali and Wohlin 2012; Martin 2002; Sidky et al. 2007). Most importantly, agile practices aim to deliver value in small increments, facilitate collaboration and anticipate changes in customer needs and the environment (Hazzan and Dubinsky 2009; Alahyari et al. 2017).

Lean software engineering is an adaptation of lean manufacturing principles in the software domain (Poppendieck and Poppendieck 2003). Lean principles are often applied in conjunction with agile practices (Rodríguez et al. 2012; Alahyari et al. 2019). Both lean and agile emphasize rapidly delivering customer value while minimizing activities that do not deliver value.

Continuous software engineering is a paradigm to deliver new software features to end-users in small increments and as rapidly as possible. The speed is achieved by combining lean and agile principles with extensive automation and removal of organizational silos (Chen 2015).

In continuous software engineering, software travels through a series of interconnected and largely automated steps to deliver the latest software changes to end-users with minimal delay. The automated steps handle testing, integration, delivery of software (Humble and Kim 2018). Once software is delivered and used by end-users, telemetry is relayed back to the software vendor to support further product decisions.

Fitzgerald et al. (Fitzgerald and Stol 2017) identifies the following components of a continuous software delivery pipeline:

1. Continuous product planning considers various inputs (customer feedback, market data, etc.) and prepares plans on leveraging software engineering to attain organizational objectives (Lin 2018; Provost and Fawcett 2013)

2. Continuous software integration comprises continuous development, configuration management, testing, integration, and other activities to produce working software (Meyer 2014; Felidré et al. 2019; Hilton et al. 2017).
3. Continuous deployment making the latest software features available for delivery to end-users (Zhu et al. 2016; Senapathi et al. 2018).
4. Continuous delivery refers to the ability to enable end-user access to the latest features, i.e. updating production software, at any time and with minimal delay (Dubey and Wagle 2007; Chen 2017; Mäkinen et al. 2016)
5. Continuous use & trust arising from less disruptive releases and improved value delivery (Gefen et al. 2003; Susarla et al. 2009).
6. Continuous feedback collecting customer feedback to support further product planning (Guzman et al. 2017; Lin 2018; Provost and Fawcett 2013; Fabijan et al. 2017)
7. Continuous improvement measures the performance of software delivery and fine-tuning the pipeline. Metrics such as cycle-time, batch size, mean-time-to-recovery, and throughput are proposed by Humble et al. (Humble and Kim 2018) to monitor the pipeline.

There are many studies focusing on the implementation, challenges, and potential solutions of individual components of CI/CD, especially the continuous integration (CI). However, we identify lack of a holistic discussion on the relevance and interrelations of all continuous practices.

For instance Hilton et al. (Hilton et al. 2016) analyzes continuous integration (CI) in open-source projects. Their findings show that the use of CI correlates with more frequent releases and popularity of open-source projects among other benefits. The main obstacles in adopting CI are lack of familiarity with CI, lack of test automation, and a slow rate of contributions to the project. Elazhary et al. (Elazhary et al. 2021) analyzes benefits and trade offs of CI practices. One of their main findings is that due to differences in implementations, contexts, and perceptions, CI cannot be studied as one practice. Rather, a more fine-grained view is needed to analyze CI.

To our best knowledge, there are no similar studies to e.g., Hilton et al. (Hilton et al. 2016) and Elazhary et al. (Elazhary et al. 2021), analyzing the suitability of continuous practices throughout the whole software life-cycle, as suggested by Fitzgerald et al. (Fitzgerald and Stol 2017). Furthermore, most existing work analyzes state-of-practice in organizations who have already adopted continuous practices to some extent. This approach creates a sampling bias and excludes organizations who have not yet overcome the adoption threshold from analysis.

Our study contributes to state-of-the-art twofold. Firstly, we analyze continuous practices, associated investments, benefits, and contextual factors from a holistic perspective. We consider the whole software life-cycle. That is, from the inception of a feature in planning stage to analyzing customer feedback on the feature and using it in fine-tuning the feature as suggested by Fitzgerald et al. (Fitzgerald and Stol 2017).

Secondly, we analyze CI/CD in two organizations in telecom domain who have not yet achieved operational level on any of the continuous practices. This perspective allow us to analyze contexts where CI/CD adoption is challenging and requires most support.

2.1.1 Cost-Benefit Analysis

The cost-benefit analysis is a method to evaluate decisions in terms of their consequences. That is, to what extent the investment (cost) in a project is justified by the gains (benefits) of the project (Drèze and Stern 1987), where the ”project” is the change required to realize the CI/CD environment. The cost-benefit analysis method is widely applicable and could be tailored to fit a variety of scenarios. The method comprises the following general steps (Sassone and Schaffer 1978):

1. Identify stakeholders and their goals concerning the project being evaluated.
2. List alternatives. It could be a decision between maintaining the current or moving to a new state or deciding between any number of alternatives.
3. Define metrics to use in the evaluation. The metrics qualify the stakeholder goals from Step #1.
4. Determine the costs and benefits associated with the project.
5. Establish a timeline of when costs are incurred and benefits reaped.
6. Express the costs and benefits in similar units. Similar units are required to perform a direct comparison between investments and benefits.
7. Calculate the net present value and ratio.
8. Perform sensitivity analysis. Determine to what extent adjustments in the inputs (investments) affect the consequences (benefits).
9. Make decisions.

To support the application of the cost-benefit analysis for evaluating the adoption of continuous software delivery practices, there is a need for an improved understanding of:

1. Common stakeholders of a CI/CD pipeline and detailing of their respective goals
2. A fine-grained understanding of investments (costs) and benefits (estimated) associated with CI/CD and its components
3. Relevant metrics to gauge the progression towards said goals
4. A methodology to quantify and compare different benefits and costs
5. A methodology to determine a configuration of continuous practices given the context of the organization

King and Schrems (King and Schrems 1978) propose using cost-benefit analysis to evaluate decisions regarding software development and operation. They identify possible types of project benefits, such as:

- Cost reduction or avoidance
- Error reduction
- Increased flexibility
- Increased speed of activity
- Improvement in management planning or control

Furthermore, they identify the following types of project costs (investments):

- Procurement costs of acquiring tools, equipment, etc.
- Start-up costs incurred due to starting the project, e.g., adopting tools and practices, training, and disruption to the organization.
- Project-related costs such as personnel, overhead, data collection, and software modifications.
- Ongoing costs, for example, maintenance of software and hardware.

The use of cost-benefit analysis to evaluate decisions in various areas of software engineering, such as adoption of cloud computing (Maresova et al. 2017; Chandra and Borah 2012), software architecture (Carriere et al. 2010), and requirements (Letier et al. 2014), is not new.

Common challenges in applying cost-benefit analysis include identifying the right metrics, expressing all costs and benefits in common units, and identifying all plausible alternatives for comparison (King and Schrems 1978). Our work aims to identify possible benefits, costs, and requirements under which an organization can realize the benefits. Such a map is the first step in aiding management and other pertinent organizational units to invest in implementing CI/CD and maximizing its operational benefits.

3 Research Methodology

3.1 Aims of the Study

This study aims to explore CI/CD from a cost (investment) and benefits perspective. Specifically, we are interested in what benefits and associated prerequisites to these benefits are linked to continuous practices. We also try to ascertain which benefits (as reported in the state-of-the-art) are considered relevant in state-of-practice (industry).

3.2 Research Questions

To attain our objective, we set forth the following research questions:

RQ1: What are the key steps of a CI/CD pipeline?

Rationale: CI/CD is an umbrella term to describe a series of mostly automated and interconnected practices. We aim to break down the concept into its constituents to support our further and more detailed analysis. We go beyond (sprint-based) engineering and include product management/planning, realization, but also delivery/commissioning and maintenance/evolution of the software-intensive product and service development.

RQ2: What potential benefits and potential investments (costs) are inherent to the CI/CD adoption?

Rationale: We aim to understand what motivates organizations to adopt CI/CD practices, what benefits motivate this, and what investments are required. We also analyze the differences between benefits triggering the adoption of CI/CD and the benefits with the most relevance

RQ3: What is needed to support the CI/CD cost/benefit analysis in the industry?

Rationale: We aim to identify opportunities for further research into the benefit/consequence analysis of continuous software engineering practices.

RQ4: What are the gaps between state-of-the-art and state-of-practice in relation to CI/CD?

Rationale: We aim to identify underdeveloped areas of CI/CD to aid further research into the area.

3.3 Research Method

We answer our research questions by conducting two industrial case studies supported by a structured literature review (not to be confused with a systematic literature review). With the case studies, we explore practitioners' needs and aims towards CI/CD. From state-of-the-art, we develop a conceptual model of continuous software delivery and use it to connect state-of-the-art to state-of-the-practice.

Both the structured review and case studies are conduced in parallel and support each other. In Fig. 1 we show an overview of our research methodology.

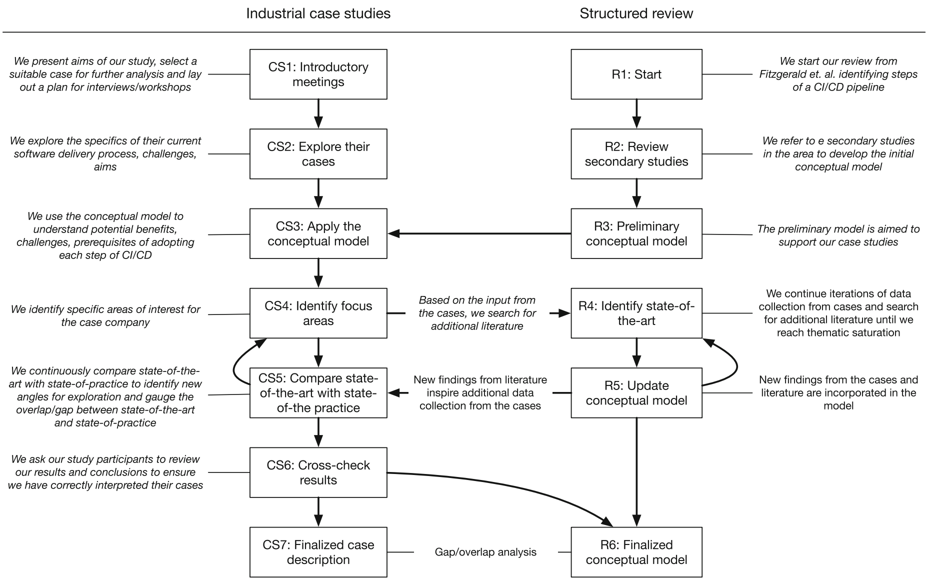
Industrial case studies
Structured review
We present aims of our study, select a suitable case for further analysis and lay out a plan for interviews/workshops
CS1: Introductory meetings
R1: Start
We start our review from Fitzgerald et. al. identifying steps of a CI/CD pipeline
We explore the specifics of their current software delivery process, challenges, aims
CS2: Explore their cases
R2: Review secondary studies
We refer to e secondary studies in the area to develop the initial conceptual model
We use the conceptual model to understand potential benefits, challenges, prerequisites of adopting each step of CI/CD
CS3: Apply the conceptual model
R3: Preliminary conceptual model
The preliminary model is aimed to support our case studies
We identify specific areas of interest for the case company
CS4: Identify focus areas
Based on the input from the cases, we search for additional literature
R4: Identify state-of-the-art
We continue iterations of data collection from cases and search for additional literature until we reach thematic saturation
We continuously compare state-of-the-art with state-of-practice to identify new angles for exploration and gauge the overlap/gap between state-of-the-art and state-of-practice
CS5: Compare state-of-the-art with state-of-the practice
New findings from literature inspire additional data collection from the cases
R5: Update conceptual model
New findings from the cases and literature are incorporated in the model
We ask our study participants to review our results and conclusions to ensure we have correctly interpreted their cases
CS6: Cross-check results
CS7: Finalized case description
Gap/overlap analysis
R6: Finalized conceptual model

Fig. 1 Overview of the research method. Alignment between case studies and the structured review

3.3.1 Industrial Case Studies

We analyze the adoption of CI/CD at two companies, Ericsson AB and Telia Company AB, using in-depth case studies and following the case study research methodology proposed by Runeson (Runeson et al. 2012). The object of our study is the continuous software engineering process, related practices, relevant organizational aims, and challenges from the benefit-consequence (cost) perspective. We limit the scope of our inquiry to one specific product in each company.

The data collection took place between September 2020 and April 2021. Each organization was involved in capturing the broad view from multiple perspectives affected by and instrumental to the area, including developers, project managers, product managers, line-manager, test engineers, and co-workers representing the operations perspectives.

We structured the interviews and workshops to familiarize ourselves with each case and their software delivery process specifics, see Step CS1 in Fig. 2. We perform inventory of their current process and continuous practices. The purpose of the inventory is to understand what practices (continuous or not) are currently applied in each case (RQ1), why, what benefits could be realized by adopting continuous practices, and what are the associated investments to realize said benefits (RQ2). Prior to the company engagements we designed a spreadsheet to keep track of the gathered data, see the structure of *Spreadsheet A* in Table 1.

We further aim to capture broader organizational goals to gauge how continuous software engineering can be useful and what parts of the pipeline are relevant to attain these goals. Looking at the broader organizational goals we identify focus areas to steer our further investigation.

We aim to complement our discussions with with document reviews, for instance process overviews, presentations outlining organizational aims, and strategies. Such materials will serve as input for steering discussions in the workshops.

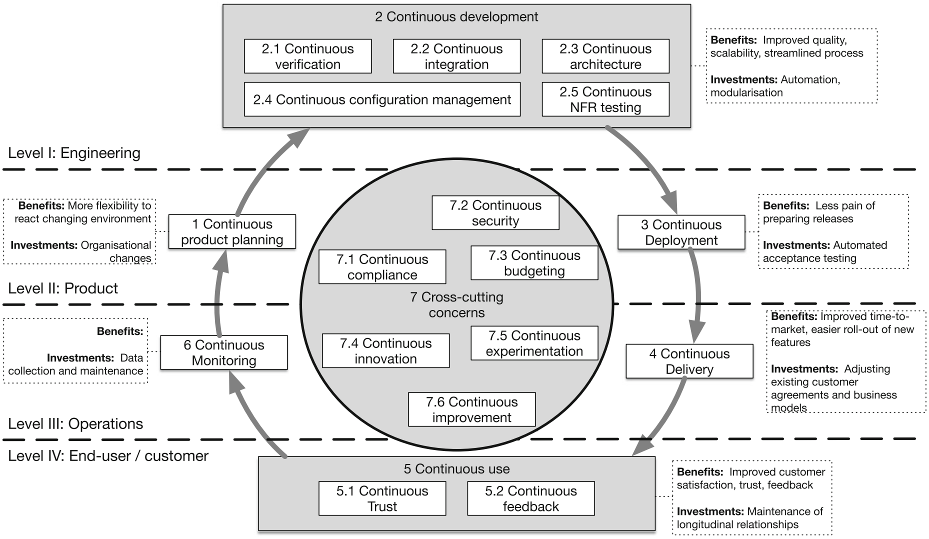
2 Continuous development
2.1 Continuous verification
2.2 Continuous integration
2.3 Continuous architecture
2.4 Continuous configuration management
2.5 Continuous NFR testing
Benefits: Improved quality, scalability, streamlined process
Investments: Automation, modularisation
Level I: Engineering
Benefits: More flexibility to react changing environment
Investments: Organisational changes
1 Continuous product planning
7.2 Continuous security
7.1 Continuous compliance
7.3 Continuous budgeting
7 Cross-cutting concerns
7.4 Continuous innovation
7.5 Continuous experimentation
7.6 Continuous improvement
3 Continuous Deployment
Benefits: Less pain of preparing releases
Investments: Automated acceptance testing
Level II: Product
Benefits:
Investments: Data collection and maintenance
6 Continuous Monitoring
4 Continuous Delivery
Benefits: Improved time-to-market, easier roll-out of new features
Investments: Adjusting existing customer agreements and business models
Level III: Operations
Level IV: End-user / customer
5 Continuous use
5.1 Continuous Trust
5.2 Continuous feedback
Benefits: Improved customer satisfaction, trust, feedback
Investments: Maintenance of longitudinal relationships

Fig. 2 The conceptual model of state-of-the-art CI/CD pipeline

As we discover new angles and perspectives, we aim to schedule additional data collection activities with additional stakeholders to capture these new perspectives. As we gain insights from the case studies, we update our conceptual model and analyze the gap/overlap between state-of-the-art and state-of-practice. With this analysis, we identified overlapping parts, gaps, and areas for further discussion.

We analyzed both cases and looked at state-of-the-art in parallel. Thus, findings from one case could be immediately cross-checked with the other case and state-of-the-art, see steps CS4–6 and R4–5. Once our analysis reached thematic saturation we finalized our reports, and presented the results to the cases for additional input and feedback.

Once our case studies reached theoretical saturation and no new concepts emerged we finalized our conceptual model. The conceptual model, see Fig. 2, is a emerged from the case studies while maintaining connection to state-of-the-art. The final iteration of the model was presented to both studied cases for additional input and feedback.

3.3.2 Connection to State-of-the-Art

To design and support the case studies, we refer to state-of-the-art. Our aim is to use state-of-the-art to establish a baseline perspective of CI/CD to support our interactions with practitioners. Specifically, we are interested to what extent benefits and investments of continuous practices from literature are relevant for the studied cases.

Our approach should not be confused with a systematic literature review or mapping as defined by e.g. Kitchenham (Kitchenham 2004) or Pettersen (Petersen et al. 2008). Rather we build upon existing secondary studies and look for papers linked in these studies to explore each step of CI/CD.

The initial scope of the review was defined by using the CI/CD research roadmap (Fitzgerald and Stol 2017). The roadmap by Fitzgerald et al. (Fitzgerald and Stol 2017) was selected as it provides the a comprehensive overview of CI/CD concepts with relevant references. The roadmap was complemented with results from a systematic literature

Table 1 The structure of *Spreadsheet A* to support inventory of continuous practices at each company

#	Heading	Description	Running example
1	Step	Step in the engineering process as as defined by the participants, e.g. Planning, Development, Testing, Maintenance, Operations	Release
2	Continuous practices	Pertinent continuous practices and their status (planned, in progress, implemented, irrelevant, not considered)	Containerization (implemented), cont. deployment (planned)
3	How it works?	Description of current implementation of the engineering step, what stakeholders are involved, triggers, exact practices, tools, routines, sub-steps	With each release we used spend a lot of time preparing installation instructions
4	Rationale	Rationale and reasoning for the current implementation, e.g. advantages, drivers	Operations does not know much about the product, they rely on our instructions and support
5	Challenges	Specific challenges with the current implementation	Despite the extensive documentation, we need to jump in and support operations with almost every installation
6	Ideal implementation	The ideal implementation of the stage given the organization context and needs	Documentation is replaced with installation scripts. We move towards DevOps
7	Benefits	Perceived or anticipated benefits from continuous practice at this step	Less time wasted on writing instructions, fewer human errors
8	Investments	Perceived or anticipated investments in relation to continuous practices at this step	Development of automation scripts, changing the current process
9	Type	How often the benefit or investment is perceived (One-off, per release, continuously)	Per release
10	Additional notes & quotes		

review (SLR) (Shahin et al. 2017). To gain a more in-depth perspective, we performed a backward snowball sampling iterations, i.e. followed the references, from the research roadmap and the SLR to find the primary studies discussing pertinent concepts, see steps R1–2 in Fig. 1.

We performed narrative analysis of the discovered papers to gain an overview of pertinent themes, associated issues and concerns relevant to our case studies (Huang et al. 2018). With the analysis we extracted practices, benefits, associated investments, challenges, and preconditions for applying the practices and realizing their benefits. Extracted data were organized in a *Spreadsheet B*, With this input we formulated the preliminary conceptual model, see R3, in Fig. 1, and prefilled the inventory spreadsheets with potentially relevant insights for discussion in the workshops. Different slices of the *Spreadsheets A and B* are presented in tables in the results section.

As the study progressed and more input from the cases was collected, see Steps CS3–5, in Fig. 1, we performed additional literature searches and thematic analysis, to support the case studies and evolve the conceptual model further, see Steps R4–5. Simultaneously, we expanded the scope of the inventory to reflect new findings from literature.

3.4 Threats to Validity

Construct Validity captures to what extent the operational measures captures the intended phenomenon (Runeson et al. 2012). A potential threat in this category stems from different understanding and terminology between researchers and industry participants. To minimize this threat, we start or interactions with industry participants with a presentation of the conceptual model. We further ask the participants to point out parts of the model known to them and single out new parts. Through such discussion, we establish a common view on the studied phenomena.

Another treat arises from our review methodology. Our aim was to gain a broad and good-enough support for conducting the case studies and to identify discrepancies between state-of-the-art and state-of-practice. There could be relevant studies that study overlooked. To mitigate this threat we divided our review per CI/CD step to more precisely target pertinent literature, and continuously compared state-of-the-art and state-of-practice to gain as broad coverage as possible.

State-of-the-art presents continuous practices spanning the entire life-cycle of a feature, from the inception of a feature idea, development, delivery to customers, use, collection of feedback, and use of feedback to steer the next iteration. However, our studied cases offer primarily focus on development and delivery steps. Therefore, there is a threat that the other steps of continuous engineering are not sufficiently covered.

Our studied cases have implemented or attempted to implement on some of the continuous practices. Thus, their input on the other practices is based on their expert opinion, not first-hand experience. This introduces a threat that they may have overlooked or misinterpreted some aspects of such practices. To counter this threat, we compare and point out differences between state-of-practice and state-of-the-art.

Internal Validity concerns the causal relationships and potential confounding factors to the explored phenomenon (Runeson et al. 2012). In this paper, we do not attempt to establish causal relationships. Thus, this category of threats is not relevant.

External Validity concerns the extent of generalizability of presented results (Runeson et al. 2012). The empirical part of our study considers two cases from the telecom domain. Thus, the generalizability of our empirical findings may be limited to similar domains and types of companies.

In reviewing state-of-the-art, we consider all sources irrespective of domain, product, and market type. Thus, our review represents a broad view of the phenomenon. We devise the proposed conceptual model from earlier systematic literature reviews. That said, the mapping between state-of-the-art and state-of-practice is limited to our two cases belonging to telecomunications domain and relatively mature companies. State-of-the-art may have additional limitations that we did not discover due to the nature of our studied cases. For these reasons, the conceptual model could be skewed towards the studied cases.

Reliability examines to what extent the data and the analysis are independent of specific researchers (Runeson et al. 2012). The first author conducted the workshops, interviews, and other data collection activities. During the data collection activities, we used our conceptual model to guide and let participants drive the discussion. Thus, the focus areas highlighted challenges, and other findings emerged independently from researchers. After data collection activities, the collected data was reviewed and discussed by both authors. These discussions helped to shape ideas and focus areas for upcoming data collection activities. During continued interactions with practitioners, we frequently discussed our intermediate findings to gain additional insights and validation. Stakeholders reviewed the final draft of the paper from both companies to ensure that the presented findings are accurate.

Another concern to the reliability of our study is replicability of the results.

4 Results

To understand drivers behind the adoption of CI/CD practices, we study state-of-practice in two companies and map our findings with state-of-the-art. Using this mapping, we highlight parts that overlap, are missing in state-of-the-art, but also items overlooked by practitioners in state-of-the-practice. We differentiate between 5 types of mapping points, see Table 2.

We structure our results in two parts. In the first part, we present two industrial cases highlighting their contexts, wants, and needs in relation to CI/CD. In the second part,

Table 2 Types of mapping points between state-of-the-art and state-of-practice

#	Type/term used in the discussion	Description
1	Confirmed	A phenomenon (a pipeline component, benefit, investment, assumption) is both described literature and implemented (or experienced, attained) by practitioners
2	Planned	A phenomenon is described by literature and anticipated by practitioners, however not yet achieved, e.g., planned but not yet implemented pipeline component.
3	Gap	A phenomenon is reported by practitioners, however not sufficiently described by state-of-the-art.
4	Irrelevant	Literature describes a phenomenon, however practitioners report it as irrelevant/not feasible/unattainable in their current context
5	Unclear	A phenomenon is described in the literature and appears potentially relevant, however additional investigation is needed to gauge whether it is relevant/attainable in practice.

we present state-of-the-art perspecitve, connect it to the state of practice to identify gaps, discrepancies, and relevant results to support the adoption of CI/CD in practice.

4.1 Case I: Ericsson

4.1.1 Research Context

Ericsson AB is a large telecommunications hardware and software provider based in Sweden. The company offers an extensive portfolio of software-intensive products. They combine their offerings with 3rd party products to create customer-specific solutions. Customer solutions are highly customized and delivered to mobile network service providers globally.

In this study, we focus on a product aimed to support mobile telecommunications operators' business functions (Product A). It is a large and complex constellation of different components. Individual software components have a varying degree of continuous integration and deployment capabilities. However, the challenge lies in the solution-level integration and testing. Differences between components, individual component delivery methods, customer-specific customizations, dependencies on third-party components, and domain regulations make final software integration complex, slow, and manual effort-intensive.

Our primary contact in Ericsson was a service delivery organization. The organization is responsible for orchestrating Ericsson's different activities to attain efficient software deliveries to customers and align the delivery process with their needs. Recently, they have started an initiative to create a strategy for using CI/CD to streamline the software delivery process, focusing on implementing a universal pipeline for delivering the Product A.

4.1.2 Data Collection

We organized the data collection activities in Ericsson between September 2020 and March 2021, see Table 3. In addition to meetings and workshops, Ericsson provided access to a large number of slide decks containing information about their ongoing work on streamlining software delivery. We used these resources to prefill our inventory spreadsheet and steer discussions in the meetings.

Due to the size of the product and large number of stakeholders involved, the inventory spreadsheet was filled in incrementally. Stakeholders were able to provide information only in their respective areas of responsibility. In one of the final workshops we gathered a larger group of stakeholders to discuss the overall picture. We also jointly developed a mind map capturing the organizational goals, links to specific engineering challenges to be solved, pertinent continuous practices, and obstacles on implementing the practices.

We summarizing the insights from Ericsson illustrating their case in the following subsections. We discuss the mapping between our empirical findings and state-of-the-art in Section 4.4.

4.1.3 Steps of the Software Delivery

The software delivery in Ericsson is a multi-stage process. It can be summarized as follows:

1. Individual components are developed using a varying degree of continuous development practices. The latest components are placed in a repository for further integration.

Table 3 Data collection activities with Ericsson

Date	Activity	Industry participants	Duration	Purpose, data collected
2020-09-08	Meeting	2	1h	Introductory presentation and discussion of the study
2020-09-09	Workshop	1	1h	Capture their CI/CD aims and strategy, inventory
2020-10-07	Interview	2	1h	A follow-up on their aims and strategy
2020-10-09	Interview	2	2h	Inventory of CI/CD benefits and challenges
2020-10-13	Workshop	2	1.5h	Relevance of CI/CD benefits in their context
2020-11-13	Meeting	2	0.5h	Workshop planning
2021-03-03	Workshop	8	2h	Capturing multiple stakeholder perspectives on software delivery issues and potential solutions, creating a mind map of orgaizational goals, challenges irt. CI/CD
2021-03-22	Meeting	1	1h	Presentation of the intermediate results, discussion

2. Service delivery organization combines different components into customer solutions. Frequently, customer solutions contain bespoke customizations.
3. Customer solutions are deployed in a staging environment where both software vendors and customers verify the software.
4. Once accepted, a solution is taken to customers' premises for further validation, configuration, and integration with customers' systems. It may take several months and up to a year until an accepted solution goes live.
5. There is a predefined update schedule determining when customers should be prepared for upgrading their solution. The interval between upgrades spans multiple years.

More often than not, transitions between steps of the software delivery are a pull rather than push operation. The reasons include ensuring control and stability of staging and operational environments and service level agreements between the vendor and customers. For similar reasons, verification of releases, especially regarding compliance and non-functional qualities, is performed manually and as a separate step before deployment.

4.1.4 Aim I - Rapid Delivery of New Technologies

Ericsson aims to remain on top of the market with the rapid delivery of innovative offerings. Emerging technologies such as 5G will require software vendors to react more rapidly to new market needs and provide innovative solutions on short notice. The current release pace is not sufficient to address this objective. Specifically, the preparation (by vendor) and adoption (by customer) of releases are overly time-consuming, complex, and expensive. As one interviewee reflected:

> *"With 5G we must be much faster in rolling out new services. The technology is going to be much more dynamic. We can't wait 6 months to deliver something and then 6 more months to get feedback."*

To address this objective, the study participants identify two areas of improvement. Firstly, rapid software delivery depends on streamlined development supported by automation—secondly, efficient solution level integration and deployment to staging environments.

Continuous development is practiced to a significant extent. Test, build, and low-level integration is largely automated already. However, due to compliance requirements and legacy processes, some verification steps are performed manually and create a bottleneck at the end of the development phase.

Solution level integration, development, and staging are implemented to deliver highly customized solutions and to jointly with a customer to verify that a release is stable and reliable for adoption. Much time is spent ensuring the reliability of the release and waiting for the proper launch window to upgrade customers' systems.

The customer systems are often piecemeal solutions consisting of various components from various vendors and in-house developments. Thus, upgrading part of a system may require upgrades and additional developments in 3rd party components. An important part of Ericsson's offering is to provide solutions that fit within existing customer systems. There is an ongoing initiative to explore to what extent the solution automation can be streamlined and automated. As an interviewee described customer solutions:

> *"Customers usually develop their systems over time and look for what is more economical for them. A typical solution contains some of our components, components from other vendors, and some of their own software. Ericsson puts in a lot of effort to make sure our components are compatible with whatever customers have built."*

Ericsson needs to implement and streamline their continuous development, deployment, and, potentially delivery, practices to fulfill this aim. However, state-of-the-art offers little support for practicing CI/CD in complex regulated environments; see rows 1–3 in Table 4.

4.1.5 Aim II - Universal Pipeline to Deliver Software

There is an aim to simplify and unify the software delivery process for different offerings and customers. The current software delivery process is tailored to the specifics of individual offerings and the needs of specific customers. Thus, there is little standardization and, consequently, much room for optimization.

Interviewees identify that the product architecture, for example, component scopes and interfaces, are not optimal for continuous solution level integration. The architectural shortcomings increase solution complexity, thus making efficient integration and delivery difficult. As described by a workshop participant:

> *"We have a "red team" of our top engineers. Initially it was planned only for incident response. However, they are involved all the time whenever we are preparing a release."*

To remedy this, Ericsson needs to evolve their architectures with CI/CD in mind. Literature suggests that cloud-native components with few interdependencies perform best. However, to what extent investments in evolving existing products to fit well with CI/CD can be justified is unknown.

4.1.6 Aim III - New Business Models

More frequent software deliveries and access to feedback could enable closer cooperation between Ericsson and customers. Thus, enabling new collaborative business models.

Table 4 Comparison of lessons from state-of-the-art and state-of-practice (Case I)

#	Aims	Lessons from state-of-the-art	Lessons from state-of-practice
1	I	Practice continuous software integration to enable continuous deployment (Humble and Kim 2018; Kim et al. 2008; Meyer 2014).	Product complexity, per-customer customizations, and dependencies to 3rd party components hinder continuous integration. Ensuring compliance and reliability of a complex product requires substantial manual effort.
2	I	Practice continuous deployment to make latest features ready for immediate delivery (Fitzgerald and Stol 2017; Chen 2015)	Latest features are enabled by customers integrating the new features in their software ecosystem. Different organizations work in different paces leading to one organization waiting for another.
3	I	Practice continuous delivery for faster time-to-market (Humble and Kim 2018; Chen 2015)	Access to customer environments for software deliveries are out of the question due to service level agreements, domain complexity, security, and reliability concerns.
4	II	Architect the product for continuous delivery (Chen 2018)	Refactoring a large existing product to stateless micro-services requires substantial effort and creates many risks.
5	II	Implement software delivery pipeline (Humble and Kim 2018)	With customer-specific process, product, and service level agreement customizations, the software vendor needs to maintain a customized pipeline for each customer.
6	III	Adopt CI/CD as the only means for software delivery (Fitzgerald and Stol 2014)	Upgrading existing customers to CI/CD is a substantial undertaking and may require supporting multiple software delivery methods. Thus, adding a lot of organizational and product complexity and cost. To what extent benefits from such upgrade exceed the costs is difficult to calculate
7	IV	Collect product usage data, telemetry, and customer feedback to steer further product direction (Humble and Kim 2018; Provost and Fawcett 2013)	Customers are not ready to share any live data to protect their business secrets and honor their agreements with end-users. Current product road-maps lack the flexibility to be frequently adjusted due to many dependencies on regulations, standards, and other road-maps.

Our study participants reflect that this is a very vague aim that needs more details to judge whether it is relevant in the telecom domain. Further discussion revealed that developing new business models must support the existing offerings until all customers are upgraded to the new models. Some customers may never accept continuous deliveries. This creates a potential situation when the organization offers the same product with continuous deliveries and plan-driven software releases. Quoting one participant:

> *"Sure, it sounds exciting! However, telecom domain has been very slow in adopting innovations in business models. Largely, because we do what operators want and some operators do not really want any innovation."*

Support for parallel business models and software delivery methods adds new requirements and complexity throughout the organization. For example, software must be developed with support for both the old and the new delivery method. Both delivery methods need to be synced to ensure consistency in terms of product features and quality. Customer support must be prepared to assist in both scenarios.

State-of-the-art assumes that the product is always delivered continuously (or release-based) and offers little support for transitioning from one model to another, see row 6, in Table 4.

4.1.7 Aim VI - Data-Driven Decision Making

Ericsson aims to benefit from data-driven techniques in improving both internal processes and customer offerings. To attain this objective, Ericsson needs to implement metrics to measure the software delivery process and establish means of getting access to product telemetry.

The study participants reveal that there have been attempts to gain access to product telemetry and usage data. However, customers are unwilling to share any data to protect their business secrets and the privacy of end-users. As one participant reflected:

> *"We have had endless discussions to get access to some telemetry. However, our customers see that as a risk rather than an opportunity."*

Ericsson has only partial control over the software delivery pipeline. The software vendor controls the pipeline internally until the latest release is ready for delivery. However, the vendor has limited control of whether customers upgrade to the latest version and no control or access to software once it is operational. One interviewee shared an old but relevant anecdote:

> *"Once in the ninethies we delivered a new solution with both hardware and software to an operator. It cost them a lot. Our sales teams tried to follow up and learn how satisfied the customer is. However, no useful information came back. Finally, six months later we sent a representative to investigate. It turned out that the operator had not even unpacked the boxes yet."*

Recently with the adoption of cloud-based solutions, the situation has improved. However, the challenge of gathering data from customers is still relevant.

4.1.8 What is Needed to Support Retrofitting Product Product A with CI/CD Pipeline?

Analysis of Case I shows that retrofitting a product with a CI/CD pipeline is a substantial undertaking and introduces many business risks in addition to technical challenges.

For example, from customers' perspective switching to a new product release model and renegotiating service agreement may trigger consideration of alternative offerings by competitors.

Comparing lessons from state-of-the-art and Case I, see Table 4, we identify the following areas for support.

Component Granularity and Flexibility Current granularity of components is not well suited to support different configurations of customer solutions. Thus, there is additional bespoke development at each release to tailor the solution to customer needs, see rows 1-4 in Table 4.

Component boundaries and limits of flexibility need to be revised to streamline the release process and keep bespoke development at minimum.

Support for Different Parallel Delivery Models Not all customers could be ready for changes in the product and its delivery method. Customers have built their own and tightly coupled solutions on top of Ericsson's offerings. Thus, any changes in architecture, components, interfaces, etc., will have a substantial impact, see rows 1–2,4–6 in Table 4.

Currently, customers rely on a predefined release pace with ample time to prepare other systems and their business processes for integration with the new release. Thus, different customers may require different release paces and extent of backward compatibility. Consequently, Ericsson must be prepared to roll out any changes in components and release pace softly while maintaining full support for customers who do not wish to change right away.

Ericsson needs to balance investments, risks, and potential benefits to decide whether it is worth the investment to change the product architecture and upgrade current customers to the improved software delivery model. We summarize the decision in Table 5.

Our interviewees state that the organization has little appetite for the risk of losing existing customers. Thus, the focus is set to adopt CI/CD with minimal disruption to customers. However, the objectives to change the software delivery model without affecting customers are contradictory. More understanding of the customers' perspective and willingness to switch to continuous software deliveries is needed.

4.2 Case 2: Telia Company

4.2.1 Research Context

Telia Company AB is a telecommunications services provider in Sweden. The company provides telecommunications services to private and business customers. To serve its customers, the company integrates in-house built software with off-the-shelf products.

Table 5 CI/CD investment, benefit, opportunity and risk conundrum in Case I

Investments of refactoring components to provide more streamlined integration of customer solutions, streamline software delivery process.	**Benefits** of faster and cheaper software releases from the vendor's perspective.
Risk of customers switching to competitors' offerings when pushed to adopt new architectures and release pace with insufficient support and leeway.	**Opportunities** to deliver new features faster and reducing the effort of accepting new releases from customers viewpoint.

The current release pace is slow due to domain complexities and dependencies on other systems with long release cycles. However, there is an ongoing initiative to improve internal efficiency and speed up software releases. The initiative is not specific to CI/CD. However, continuous software delivery is considered to be one of the candidate solutions.

The company had selected one of their products (from now on, Product B) as a pilot case for adopting continuous engineering practices. The product is a software layer between other systems supporting an exchange of text messages. The product is stable, mature, and most developments concern maintenance updates and occasional bug fixes. The product is not exposed to customers directly. Instead, it enables features for other products with user interfaces.

4.2.2 Data Collection

We conducted several workshops with the product team to understand their objectives, motivation, constraints, and challenges in adopting CI/CD practices. The product manager, two senior developers, delivery manager, and operations specialist where the core participants. The data collection took place between September, 2020 and April 2021, see Table 6.

4.2.3 Steps of Software Delivery

Product B is relatively small and developed by a loose team of few developers. Other stakeholders include an operations team responsible for deploying and ensuring the smooth operation of the product. The steps of delivering Product B are:

1. Developers receive issue tickets from the operations team and implement necessary changes. The flow of tickets is slow, and developers prepare two releases a year.
2. Finished software is complemented with extensive instructions for deployment and operation and forgo several rounds of testing.
3. Once a release is ready, operations teams take over and deliver the release to customers with support from the development team.

4.2.4 Aim I - Simplify the Development and Build Process

Due to strong coupling with other systems, Product A requires a complex environment for development and testing. Setting up the local environment and ensuring the correct configuration for development and testing is a complicated task. The team aims to benefit from containerization, build scripts, and similar practices to alleviate software dependency management issues. As one participant summarized the challenge:

> *"It used to take about two weeks to set up a development environment from scratch. The product has many intricate dependencies that are documented nowhere. Most of the time is spend on troubleshooting compatibility issues. With Docker containers we do not have this issue anymore."*

The slow-release pace of two releases per year has created an issue of knowledge preservation. Over long periods of inactivity on Product B, developers are assigned to other tasks or may have left the company. Every small change requires context switching and re-learning the specifics of the product. As a consequence, developers spend substantial time reading documentation and setting up individual development environments.

Table 6 Data collection activities with Telia

Date	Activity	Industry participants	Duration	Purpose, data collected
2020-09-03	Workshop	9	2h	Introductory presentation and discussion of the study
2020-09-03	Workshop	2	2h	Introduction into product/domain specifics
2020-09-09	Meeting	3	0.5h	Planning of workshops, setting focus areas
2020-09-17	Workshop	8	2h	Inventory of their CI/CD practices
2021-01-14	Workshop	5	2h	Capture and break down their organizational aims irt. CI/CD
2021-01-25	Meeting	1	0.5h	Presentation of the intermediate results
2021-04-26	Meeting	7	2h	Workshop to explore CI/CD from customers perspective

4.2.5 Aim II - Reduce the Complexity of Software Upgrades

Currently, every release of Product B is complemented with extensive documentation. Preparing this documentation is a substantial undertaking. The exact audience and value of the documentation are unclear to the development team. However, some documentation (e.g., installation instructions, test cases) is required as software artifacts are passed from development to operations. As described by one interviewee:

> *"Deployment of the product is not straightforward and operations often need our support to get things going even with all the documentation we provide. I think with automated scripts and containers we can make deliveries easier and leave less room for issues."*

The software vendor aims to streamline software delivery process by tearing down organizational silos and closer collaboration between development and operations. Such a move should reduce the need for extensive documentation and streamline the development-operations process as there would be one team responsible for the whole process.

Current customer agreements determine the software delivery process and do not support frequent releases. Furthermore, customers are not eager to receive frequent software deliveries. They mostly contain maintenance updates with little perceived value and increase the risk of system outages and other issues. The software vendor aims to simplify the release adoption process from the customers' viewpoint. One participant reflected on this:

> *"The upgrades does not provide much new features, thus there is little incentive to upgrade right away. We have to send a notice weeks in advance to let customers prepare for an upgrade. We know very little what it takes for them to upgrade."*

4.2.6 Aim III - Improve Software Quality

Currently, Product B forgo several rounds of testing in different test environments before a release. Although some tests are automated, there is considerable potential to benefit from automated testing and verification practices.

Some tests are difficult to automate because the tests are complex, expensive to run, and prone to distrust in automated testing practices. These tests remain as a bottleneck at the end of the otherwise automated integration and verification pipeline.

Due to the high turnaround of developers, the product had eroded and accumulated some technical debt. To support the refactoring effort, the team wishes to benefit from automated and continuous verification practices.

The software vendor aims to benefit from continuous integration and deployment practices to remove bottlenecks of manual testing, simplify the release process, and improve the overall quality of the product. As stated by one interviewee:

> *"We have started automating some tests and it shows results. However, we still have some pretty heavy manual tests at the end. It would be good to automate them as well. However, we feel that sometimes there is distrust in automated tests and stakeholders are more confident with manual testing."*

4.3 What is Needed to Support Retrofitting Product B with a CI/CD Pipeline?

Currently, there are no inherent obstacles to retrofitting Product B with a continuous integration pipeline. However, the team reflects that by adding new practices and ways of working, they need to revise and remove any obsolete activities. For example, multiple testing rounds on different environments are no longer relevant if the product is containerized. Similarly, deployment scripts can replace installation instructions.

Identification of obsolete steps in development, integration, and deployment is essential to avoid unnecessary investments in automation. However, changes in the software delivery process need to be communicated and explained to all stakeholders.

Our interviewees admitted that they mainly consider the engineering perspective. The interests of other stakeholders, namely customers and adjacent products, have not been considered. Further work is needed to understand software deliveries from customers perspective and to what extent continuous software deliveries reduces the effort to adopt a new software release.

Comparing reflections from participants with state-of-the-art, see Table 7, we observe that practice agrees with literature. Specifically, continuous practices reduces the complexity of software engineering and removes tedious setup and configuration steps, and reduces the need for detailed documentation.

Continuing adoption of CI/CD, needs to explore the interests of other stakeholders. For example, exploring the potential of aligning software delivery pipelines across the whole organization. Thus, creating a coherent software delivery pipeline that can be further extended to customers organizations. We summarize the investment-benefit conundrum in Table 8.

The CI/CD initiative in the Product B was intended to pilot continuous software delivery within the organization to gauge its further potential. Thus, attaining more than "just enough" continuous capabilities could be a worthy investment to gain a complete picture of the potential benefits, investments and pitfalls of CI/CD in their specific context.

4.4 Connection to State-of-the-Art

A substantial amount of research addresses individual building blocks of a CI/CD pipeline, e.g., test automation (Wiklund et al. 2017; Raulamo-Jurvanen et al. 2017; Kasurinen et al. 2010). However, a big picture perspective is needed to identify all relevant components of a pipeline (RQ1) and understand how individual components of a pipeline fit together to analyze their benefits (RQ2) (Frank 2000).

Table 7 Comparison of lessons from state-of-the-art and state-of-practice (Case II)

#	Aims	Lessons from state-of-the-art	Lessons from state-of-practice
1	I	Practice continuous integration to improve software deliveries (Shahin et al. 2017)	Small and local improvements of automation and tooling can substantially simplify otherwise complex configurations without implementing full continuous integration.
2	I, II	Identify and maintain knowledge artifacts to support knowledge distribution(Ouriques et al. 2019)	Build and configuration scripts are concise and up to date product documentation.
3	II	Make software deliveries more efficient by tearing down boundaries between development and operations (Zhu et al. 2016)	Tearing down boundaries reduces the need for documentation, thus simplifying the release process.
4	II	Unnecessary steps in software engineering are waste (Poppendieck et al. 2011)	When transitioning to continuous development and integration, existing practices and ways of working need to be evaluated. Steps that do not provide value should be eliminated.
5	III	Develop an automated test framework to ensure the quality of software (Shahin et al. 2017)	Robust test framework helps to preserve product knowledge and works as a safety net to mitigate risks of refactoring and sporadic development.

Table 8 CI/CD investment, benefit, opportunity and risk conundrum in Case II

Investments in automation, tooling, streamlining software delivery processes, reorganization	**Benefits** of local knowldege preservtion, efficiency, and waste minimization
Risk of wasting resources on attaining irrelevant software delivery capabilities	**Opportunities** to free-up resources for more value adding tasks, and potentially increasing the appetite for new features; showcase the CI/CD to other parts of the organization and leading the change towards more efficient software deliveries.

Inspired by Fitzgerald et al. (Fitzgerald and Stol 2017), we develop a conceptual model visualizing state-of-the-art CI/CD pipeline, see Fig. 2. Importantly, the model is based on state-of-the-art in the area. We use it as an overview and tool in our industry workshops to aid discussions of costs, benefits, relevance, and dependencies between different parts of the pipeline.

In the figure, we denote the main pipeline steps with rectangular blocks. With dashed boxes, we illustrate the key benefits and investments associated with each step. The flow through the pipeline is denoted with arrows. Components of the pipeline are organized by the four main levels of stakeholders in a CI/CD pipeline:

- *Level I: Engineering organization* produces software. The primary objective of an engineering organization is to efficiently, in terms of time and resources, produce software according to the product planning and other organizational objectives and constraints (Fitzgerald and Stol 2014).
- *Level II: Product planning organization* analyzes various inputs and sets objectives for further product development. The planning organization's primary objective is to prepare plans on leveraging software engineering to attain strategic objectives, e.g., market share, profitability, and outperform competition (Fitzgerald and Stol 2014; Humble and Kim 2018).
- *Level III: Operations organization* deploys and provides the software to the end-users. The primary objective of product operations is to deliver quality service to end-users (Fitzgerald and Stol 2014).
- *Level IV: End-user organization* uses and benefits from the software. The primary objective of this stakeholder is to maximize perceived value from the software (Fitzgerald and Stol 2014; Boehm 2003).

These stakeholders can be organized into different organizational structures. The organizational configuration is important because more involved organizations imply more boundaries, need to synchronize different paces, aiming for different objectives, unclear areas of responsibility, and other potential impediments to end-to-end continuous software delivery (Serrat 2017; Romano Jr et al. 2010).

The conceptual model shows an idealized state-of-the-art scenario. Analysis of the two industrial cases shows that adoption of the pipeline components varies, see Table 9. We elaborate details of the cases in Sections 4.1 – 4.2. Further sections elaborate state-of-the-art of each component.

4.4.1 Continuous Planning

Continuous planning, see Step #1, in Fig. 2, refers to a holistic activity involving multiple stakeholders to create lightweight, dynamic, and open-ended product plans. The planning

Table 9 Mapping between the steps of CI/CD, state-of-the-art, and state-of-practice

#	Component	References	Case I	Case II
1	Continuous planning	(Fitzgerald and Stol 2014; Lehtola et al. 2009; Provost and Fawcett 2013)	Irrelevant	Irrelevant
2	Continuous development	(Humble and Kim 2018; Tómasdóttir et al. 2017; Williams et al. 2003; Stolberg 2009; Shahin et al. 2017; Jiang et al. 2017; Kim et al. 2008; Hasselbring and Steinacker 2017; Del Rosso 2006; Riaz et al. 2009; O'Connor et al. 2017; Hilton et al. 2016; Hilton et al. 2017)	Confirmed	Confirmed
3	Continuous deployment	(Fitzgerald and Stol 2017; Neely and Stolt 2013; Feitelson et al. 2013)	Planned	Planned
4	Continuous delivery	(Chen 2015; Humble and Kim 2018; Claps et al. 2015; Kuula and Haapasalo 2017; Shahin et al. 2017)	Unclear	Planned
5	Continuous use	(Chen 2015; Gefen et al. 2003; Susarla et al. 2009; Yaman et al. 2016)	Confirmed	Confirmed
6	Continuous monitoring	(Ehlers et al. 2011; van Hoorn et al. 2009; Olsson and Wnuk 2018; Johnson et al. 2005)	Irrelevant	Confirmed

is aimed to swiftly address and adjust to new market opportunities, changes in a business environment, and technologies (Fitzgerald and Stol 2014; Lehtola et al. 2009).

Continuous planning depends on access to immediate customer feedback and product telemetry to support decision making and organizational flexibility to make necessary adjustments rapidly, see Table 10. Without access to information, the planning activity falls short of delivering precise responses. Lack of organizational flexibility hinders the implementation of the plans as they may become outdated before the organization is able to react (Provost and Fawcett 2013; Li et al. 2010).

The benefits of continuous planning concern faster response time to any inputs, thus helping the organization to be more flexible, see Table 11.

Investments in continuous planning concern the costs of more frequent analysis, planning, and coordination, see Table 12, and investments in data collection and analysis, see Table 26.

Table 10 Assumptions associated with continuous planning

#	Assumption	References	Case I	Case II
1	Customers and end-users are ready to provide detailed feedback and share product usage data (telemetry) as input for planning	(Isaak and Hanna 2018; Zhang 2018)	Irrelevant	Confirmed
2	The release cycles and time-to-feedback are short enough to provide continuous input for planning the next cycle	(Humble and Kim 2018)	Irrelevant	Irrelevant
3	The organization is flexible to quickly adjust any plans on short notice	(Provost and Fawcett 2013; Li et al. 2010)	Irrelevant	Unclear

Table 11 Benefits associated with continuous planning

#	Benefit	Beneficiary	References	Case I	Case II
1	Less wasted resources on building irrelevant features	Organization	(Claps et al. 2015; Fabijan et al. 2017)	Irrelevant	Irrelevant
2	More flexibility from shorter planning cycles	Organization	(Lohan 2013; Lehtola et al. 2009)	Irrelevant	Irrelevant
3	Improved, data-driven decision making	Organization	(Provost and Fawcett 2013; Fabijan et al. 2017)	Irrelevant	Irrelevant

4.4.2 Continuous Development

Continuous development, see Step #2 in Fig. 2, comprises activities to produce software. Development starts by receiving plans or tasks from the product planning organization, turning these plans into working software, and returning a ready-to-be-deployed software.

The continuity of development is achieved by minimizing any waiting in the process, for example, downtime until test results are in or shelving completed features until specified release date. Another important characteristic of continuity is to minimize dependencies between tasks and developers, enabling scalability through parallelization of development tasks (Humble and Kim 2018).

The literature identifies several activities that constitute continuous development.

Continuous Verification see Step #2.1, in Fig. 2, refers to running various test suites frequently and in parallel with development. Practices such as using code linters (Tómasdóttir et al. 2017), following test-driven development, and running unit tests (Williams et al. 2003; Stolberg 2009) help to reduce the time between a defect are introduced and discovered. Source code analysis tools and pull-request review process help to ensure that the code conforms to the best practices and organizational standards (Shahin et al. 2017; Jiang et al. 2017)

Continuous integration see Step #2.2, refers to frequent integration of software components and running of higher-level tests to ensure software as a whole work as intended and enabling continuous deployment, see Step # 3, (Pinto et al. 2018; Kim et al. 2008; Meyer 2014; Hilton et al. 2017; Hilton et al. 2016; Felidré et al. 2019).

Continuous Architecture see Step #2.3, refers to a frequent assessment of software architecture to control software decay and adjusting the architecture to suit new and evolving scenarios (Del Rosso 2006; Riaz et al. 2009). The right software architecture is essential to support other CI/CD activities. For example, modularization of software reduces the need

Table 12 Investments associated with continuous planning

#	Investment	References	Case I	Case II
1	Cost of more frequent planning and coordination	(Humble and Kim 2018)	Irrelevant	Irrelevant
2	Cost data analysis to support decision making	(Provost and Fawcett 2013)	Irrelevant	Irrelevant

Table 13 Assumptions associated with continuous development

#	Assumption	References	Case I	Case II
1	Product architecture is modular and permits permits fast and independent modification, building, testing, and deployment of individual components	(Balalaie et al. 2016; Sturtevant 2017)	Irrelevant	Confirmed
2	Software development teams are independent and cross-functional enabling parallelization of development tasks	(Humble and Kim 2018)	Unclear	Irrelevant
3	Software test, build, integration and quality assurance steps can be automated to a significant extent	(Felidré et al. 2019; Senapathi et al. 2018; Pinto et al. 2018)	Unclear	Planned

to coordinate between teams working on different parts of the software, simplifies verification, and enables quick deployment, Steps 3-4 in Fig. 2, of a part of the system (O'Connor et al. 2017; Humble and Kim 2018; Chen 2018).

Continuous Configuration Management see Step #2.4, refers to a set of practices to ensure that all assets and routines for building and running software are versioned and scripted. The practices include source code versioning, isolating software instances and their dependencies in containers, build scripts, automated deployment to dedicated build/staging environment, and so on (Hasselbring and Steinacker 2017).

Continuous Non-functional Testing see Step #2.5, refers to frequently verifying non-functional aspects of software (Yu et al. 2020).

State-of-the-art identifies numerous benefits of continuous development. Most benefits arise from implementing a robust and automated process for software verification. Full automation requires software to follow certain enforced standards (such as modularization, testability, unit test coverage). As a result, the overall software quality increases, and less manual effort is needed to make sure software works as intended. Thus, providing engineers with the flexibility to dedicate their time to more value-adding activities, see Table 14.

Beneficiaries include developers who experience increased productivity and support for their tasks, the team experiencing improved culture and collaboration, product improving internal and external quality aspects, and the organization.

On the investments side, there are substantial investments to create automated test suites, integration environments, tooling, and test data. However, state-of-the-art offers limited perspective into estimating the costs of test automation and test data management, see Table 15.

Continuous development relies on test automation. Thus the ability to automate most, if not all, QA steps is a prerequisite, see Table 13.

4.4.3 Continuous Deployment

Continuous deployment, see Step #3 in Fig. 2 refers to a practice to continuously deploy the latest software to a staging environment and keep it ready for immediate delivery. The continuous deployment follows continuous development when software is integrated and passes all tests (Fitzgerald and Stol 2017).

Table 14 Benefits associated with continuous development

#	Benefit	Beneficiary	References	Case I	Case II
1	Reduced time between defect introduction and discovery	Developer	(Williams et al. 2009; Humble and Kim 2018)	Confirmed	Planned
2	Benefit of offloading heavy integration/build tasks from developer machines to dedicated integration environment	Developer	(Hilton et al. 2016)	Confirmed	Confirmed
3	Safety harness and smoother learning curve for novice contributors	Developer	(Vasilescu et al. 2015; Humble and Kim 2018; Williams et al. 2009)	Confirmed	Confirmed
4	Improved code understandability as unit tests doubles as documentation	Developer	(Williams et al. 2009)	Confirmed	Confirmed
5	Improved collaboration	Team	(Williams et al. 2009; Humble and Kim 2018)	Confirmed	Irrelevant
6	Reduced time to resolve integration issues	Team	(Lacoste 2009; Humble and Kim 2018; Rogers 2004)	Unclear	Confirmed
7	Fosters culture of individual responsibility to quality and speed	Team	(Feitelson et al. 2013)	Confirmed	Planned
8	More streamlined development process due to automation	Team	(Chen 2015; Humble and Kim 2018; Kumar and Mishra 2016; Vasilescu et al. 2015)	Confirmed	Confirmed
9	Time and resource savings from automating repetitive tasks	Team	(Williams et al. 2009; Shamshiri et al. 2015; Vasilescu et al. 2015)	Confirmed	Confirmed
10	Enforced code quality standards	Product	(Humble and Kim 2018; Vasilescu et al. 2015; Williams et al. 2009; Chen 2015)	Confirmed	Planned
11	Increased product quality	Product	(Williams et al. 2009; Kumar and Mishra 2016)	Confirmed	Confirmed
12	Improved scalability of development organization	Organization	(Feitelson et al. 2013; Humble and Kim 2018; Shahin et al. 2019)	Irrelevant	Planned

Table 15 Investments associated with continuous development

#	Investment	References	Case I	Case II
1	Development, verification, and maintenance of automated test suites	(Sundelin et al. 2018; Lam et al. 2019)	Confirmed	Confirmed
2	Preparation and maintenance of test data	Gap	Confirmed	Confirmed
3	Cost of adopting new tools and practices	Gap	Confirmed	Confirmed
4	Cost of prioritizing and executing tests	(Rogers 2004; Memon et al. 2017)	Confirmed	Confirmed
5	Preparation and maintenance of test environments	(Rogers 2004)	Confirmed	Confirmed
6	Cost of managing dependencies other pipelines/3rd party components	(Claps et al. 2015)	Confirmed	Confirmed
7	Cost of refactoring the product to support CI/CD	(Del Rosso 2006; Riaz et al. 2009)	Confirmed	Irrelevant

In the literature, we found that terms delivery and deployment are sometimes mixed or used interchangeably, see, for example, Fitzgerald et al. (Fitzgerald and Stol 2014), and Neely et al. (Neely and Stolt 2013). We differentiate between the two. We use the term deployment to refer to internal readiness to deliver software to customers at any moment. With the term delivery, we refer to the ability to deliver the latest software to end-users at any time.

Continuous deployment relies on continuous integration and test automation at the earlier continuous development step, see Table 16.

The main benefits of continuous deployments stem from their frequent and incremental nature. Minor changes are easier to verify than large updates, reducing the risk of introducing severe issues. Automation reduces stress and overtime of preparing a release, thus increasing developers' job satisfaction, see Table 17.

The investments of continuous deployment concern the development of a robust acceptance test suite, and adjusting the organization. Continuous deployment requires downstream stakeholders (operations, customer support, marketing, etc.) to work with a versionless and continuously evolving product. This requires additional coordination effort, see Table 18.

4.4.4 Continuous Delivery

Continuous delivery, see Step #4, in Fig. 2, refers to making the latest features available to end-users immediately, i.e., with automated software upgrades. The critical difference between release-based and continuous delivery is that completed features are shelved until the scheduled release date in a release-based process. While shelved, a feature does not generate value (e.g., profit for the vendor and value for the customer), and it is uncertain to what extent the feature is relevant in the market. However, in continuous delivery, completed

Table 16 Assumptions associated with continuous deployment

#	Assumption	References	Case I	Case II
1	The organization applies continuous development practices	(Fitzgerald and Stol 2017)	Planned	Planned

Table 17 Benefits associated with continuous deployment

#	Benefit	Beneficiary	References	Case I	Case II
1	Improved work-life balance	Developer	(Neely and Stolt 2013; Humble and Kim 2018)	Unclear	Unclear
2	Narrower test focus	Team	(Neely and Stolt 2013; Feitelson et al. 2013)	Unclear	Irrelevant
3	Reduced effort and stress to prepare releases	Team	(Feitelson et al. 2013; Neely and Stolt 2013; Humble and Kim 2018)	Planned	Planned
4	Reduced risk of introducing major issues due to small incremental releases	Product	(Claps et al. 2015; Humble and Kim 2018)	Planned	Planned
5	Latest features always available for delivery	Organization	(Feitelson et al. 2013; Neely and Stolt 2013; Humble and Kim 2018)	Planned	Planned

features forgo automated integration, verification, and acceptance steps are immediately delivered to the end-users. Thus, new features start generating value and feedback to steer further product development (Humble and Kim 2018; Chen 2015).

Continuous delivery assumes that the software vendor has access to the product for upgrades, and customers are willing to upgrade without notice, see Table 19. Renegotiating agreements with customers and establishing control over production environments are the key investments in adopting continuous delivery, see Table 21.

The primary benefits of continuous delivery stem from pushing the latest features to customers with no delay. Thus, increasing the value of the features and starting to collect customer feedback. In Table 20, we summarize the benefits from literature.

4.4.5 Continuous Use

The shift from transactional, release-based software deliveries to continuous, versionless software highlight the importance of delivering software that satisfies customer needs over time, not just at the moment of purchase. Business models, such as software-as-a-service, monetize software, i.e., continuous use, not just its purchase. These developments create opportunities for software vendors to establish synergy with customers, improve understanding of their needs, and build trust. Mutual trust enables organizations to open up to each other (e.g., by sharing details on how the product is used and jointly developing new experimental features) (Fitzgerald and Stol 2017; Susarla et al. 2009).

Table 18 Investments associated with continuous deployment

#	Investment	References		
1	Development of a robust acceptance test suite	(Neely and Stolt 2013; Shahin et al. 2017; Lam et al. 2019)	Confirmed	Planned
2	Revising marketing strategies to promote versionless product	(Claps et al. 2015)	Planned	Planned
3	Additional coordination effort between development/maintenance/operations/support teams	(Claps et al. 2015)	Confirmed	Planned

Table 19 Assumptions associated with continuous delivery

#	Assumption	References	Case I	Case II
1	Product is not in a regulated domain, safety, or mission critical	(Shahin et al. 2017)	Confirmed	Confirmed
2	Customers are ready to accept continuous deliveries	Gap	Irrelevant	Planned
3	Software vendor has access to the product for upgrades after decommissioning	Gap	Unclear	Confirmed

The primary benefits from continuous use arise from closer, longitudinal relationships with customers and end-users. The closer relationship enables more opportunities for feedback collection, builds trust, and improves overall customer satisfaction, see Table 23.

The investments of continuous use concern costs associated with maintaining longitudinal customer relationships and analyzing customer feedback. We differentiate between customer feedback arising from the use of the product, e.g., social media posts and app reviews, revealing experiences with the software, and product telemetry (discussed in the following subsection) capturing how the software is used, see Table 24

The benefits from continuous are relevant if the product offers features that are intended for continuous use. Software that is used rarely, e.g., a system that is used once a year to generate a yearly report, may not generate meaningful feedback or customer trust, see Table 22.

4.4.6 Continuous Monitoring

Continuous monitoring, see Step #5 in Fig. 2 is a practice to collect and analyze data on how the product is used and performs in a live environment, including metrics from the CI/CD pipeline. This data used in planning, monitoring helps to fine tune the product, CI/CD procedures, discover defects and quality issues early (Ehlers et al. 2011; van Hoorn et al. 2009)

There are no immediate benefits from collecting the data. Data collected at this step of the pipeline enables and supports continuous planning, and improvement activities, see Steps #1 and #7.6 in Fig. 2 (Olsson and Wnuk 2018; Johnson et al. 2005). Product usage data helps to prioritize test cases and to synthesize realistic test data in the continuous verification step, see Step #2 in Fig. 2, (Anderson et al. 2019).

Monitoring and data collection are associated with investments in setting up and maintaining relevant infrastructure, see Table 26. However, to implement continuous monitoring, customers should be ready to provide access to product usage data, see Table 25.

Table 20 Benefits associated with continuous delivery

#	Benefit	Beneficiary	References	Case I	Case II
1	Improved time-to-market for new features	Organization	(Chen 2015; Humble and Kim 2018)	Irrelevant	Planned
2	Faster time-to-feedback	Organization	(Claps et al. 2015; Humble and Kim 2018)	Planned	Irrelevant
3	Easier adoption of new releases	Customer	Gap	Planned	Planned

Table 21 Investments associated with continuous delivery

#	Investment	References	Case I	Case II
1	Adjusting contractual arrangements with customers, transferring customers to continuous software deliveries	(Yaman et al. 2016)	Unclear	Planned
2	Adjusting the business model to offer versionless software	(Kuula and Haapasalo 2017; Loebbecke and Picot 2015)	Unclear	Planned
3	Technical arrangements with customers	(Shahin et al. 2017)	Unclear	Irrelevant
4	Delivery infrastructure	(Shahin et al. 2017)	Unclear	Irrelevant

4.4.7 Cross-Cutting Concerns

Some pipeline components are rather cross-cutting than a discrete step in the software delivery pipeline, see components #7.* in Fig. 2. The benefits of these cross-cutting components include enable and support the rest of the pipeline, e.g. by generating data, setting performance indicators, and ensuring compliance to relevant regulations (Chen 2015; Shahin et al. 2017).

Primary investments in cross-cutting concerns tearing down existing organizational structures, business models, and ways of working to implement continuous software delivery pipeline throughout the organization, see Table 27.

Continuous Compliance see Step #7.1, in Fig. 2, refers to the practice to ensure compliance to relevant regulations continuously and throughout the pipeline, in contrast to compliance verification bottleneck at the end of development phase (Fitzgerald and Stol 2014; Moyon et al. 2018). Furthermore, agile and continuous principles are often at odds with regulatory practices. Practicing continuous software delivery in regulated environments requires a careful balance between speed and discipline (McHugh et al. 2013; Fitzgerald et al. 2013)

Continuous Security see Step #7.2, in Fig. 2, refers to practicing security throughout the pipeline (Moyon et al. 2018; Dännart et al. 2019)

Continuous Budgeting see Step #7.3, in Fig. 2, highlights the need for the whole organization to adopt continuous practices. Budgeting traditionally results in yearly, quarterly, etc., budgets tied to attaining specific objectives delegated to specific organizational units. However, such a plan-driven approach may hinder cooperation, speed, and flexibility associated with continuous practices (Frow et al. 2010; Lohan 2013).

Continuous Innovation see Step #7.4, in Fig. 2, refers to a process throughout the pipeline to respond to evolving market conditions (Cole 2001; Olsson et al. 2012).

Table 22 Assumptions associated with continuous use

#	Assumption	References	Case I	Case II
1	Product offers features encouraging continued use	(Gustafsson et al. 2005)	Confirmed	Confirmed

Table 23 Benefits associated with continuous use

#	Benefit	Beneficiary	References	Case I	Case II
1	Improved customer satisfaction, trust	Customer	(Chen 2015; Gefen et al. 2003)	Confirmed	Confirmed
2	Rich feedback from on-line communities, social media, user forums etc.	Product	(Guzman et al. 2017; Genc-Nayebi and Abran 2017; Yaman et al. 2016)	Irrelevant	Irrelevant
3	Benefit of emotional and habitual connection between customers and the use of the product	Organization	(Rodríguez et al. 2017; Gefen et al. 2003)	Confirmed	Confirmed

Table 24 Investments associated with continuous use

#	Investment	References	Case I	Case II
#	Cost of maintaining longitudinal customer relationships	(Ryals 2005)	Confirmed	Irrelevant
#	Cost of collection and analysis of customer feedback	Gap	Confirmed	Confirmed

Table 25 Assumptions associated with continuous monitoring

#	Assumption	References	Case I	Case II
1	Customers are ready to share product usage data	Gap	Irrelevant	Confirmed
2	The product planning organization is prepared to use the data and practice continuous planning	(Lin 2018)	Irrelevant	Planned

Table 26 Investments associated with continuous monitoring

#	Investment	References	Case I	Case II
1	Cost of implementing monitoring features in the software	(van Hoorn et al. 2009)	Confirmed	Confirmed
2	Cost of data storage and maintenance	(Gardner 1998; Yaman et al. 2016)	Unclear	Confirmed
3	Cost of preprocessing, cleaning the data from sensitive or bogus information	(Fatima et al. 2017; Rahm and Do 2000)	Unclear	Unclear
4	Cost of getting customer agreement to share data from the product use	(Isaak and Hanna 2018; Zhang 2018)	Confirmed	Confirmed

Table 27 Investments associated with cross-cutting concerns

#	Investment	References	Case I	Case II
1	Cost of implementing organizational changes, removing internal boundaries, adjusting business models etc. to implement end-to-end CI/CD	(Humble and Kim 2018; Chen 2015; Shahin et al. 2017)	Confirmed	Unclear
2	Cost of maintaining multiple product or customer specific software delivery pipelines, e.g. continuous and release based	Gap	Confirmed	Unclear

Continuous Experimentation see Step #7.5, in Fig. 2, refers to a controlled, data-driven process of devising hypotheses about user preferences, setting experiments, and analyzing the results to drive product decisions (Fagerholm et al. 2017; Bosch 2012).

Continuous Improvement see Step #7.6, in Fig. 2, arises from Lean principles and aims to minimize waste and perfecting the process. The improvements are achieved by process mapping, collecting metrics, observing performance indicators, and making minor adjustments throughout the pipeline (Cole 2001; Poppendieck et al. 2011).

5 Discussion

We have analyzed two cases to understand how they adopt CI/CD and their perspectives on the investments and benefits of CI/CD practices.

Our results show that in both cases, the adoption of CI/CD practices started by engineering teams wishing to improve internal efficiency. Teams had adopted automated test, build, and integration practices to aid their day-to-day work. There are immediate benefits of adopting such practices confirmed both by our cases, see Sections 4.1.4, 4.2.4, and literature, see Section 4.4.2. However, in the cases there are no systematically collected objective data reflecting the gains.

Both studied organizations report that implementing continuous deliveries to customer environments would be a significant challenge. Existing customer agreements are designed around planned releases and do not support continuous software deliveries. Deliveries to customer environments require adhering to customer-specific differences and dependencies. Furthermore, pushing customers to accept new terms of service create a risk of customers switching over to competitors' offerings. The challenge is further exacerbated by the fact that both companies offer software to other organizations to provide services for end-users. The limited control over the software delivery pipeline hinders the speed of software delivery and limits exchange of data and feedback, especially in Case I, see Section 4.1.6.

Humble et al. (Humble and Kim 2018) argues that delivering minor frequent incremental updates alleviates the release pain. However, our interviewees responded that while they find the statement plausible, they lack concrete arguments and leverage to force their customers to accept continuous software deliveries. A specific challenge arises from customers' developments on top of the vendor's software. Any update introduces a risk of breaking the intricate dependencies. For this reason, accepting software updates involves extensive testing and custom development on the customer's side. There is a potential to reduce the risk by synchronizing the development cycles between the customer and the vendor and strictly defining the interfaces. However, persuading many other downstream organizations to change and align their processes with the software vendor is unrealistic in the foreseeable future, see Sections 4.1.8 and 4.3.

Both organizations have chosen to adopt CI/CD due to ongoing high-level initiatives to improve software delivery. However, they have performed only a superficial analysis of what exact goals they wish to achieve and whether CI/CD is the most suited approach. More in-depth analysis is needed to understand how the implementation of CI/CD can support organizational objectives.

We continue the discussion by answering our research questions.

5.1 What are the Key Steps of the CI/CD Pipeline?

Literature suggests that an end-to-end continuous software delivery pipeline comprises planning, engineering, deployment, delivery, use, feedback collection, analysis, and other steps, see Fig 2 and Section 4.4. However, to our best knowledge, a complete implementation of an end-to-end CI/CD pipeline is yet to be demonstrated by empirical studies. State-of-the-art discusses parts of the pipeline and primarily focus on development, integration, and delivery steps.

In our studied cases, the scope of the CI/CD is limited to engineering, integration, deployment, and delivery steps. Practices such as continuous data collection, planning, and improvement are appealing, however currently unattainable, see Sections 4.1.8 and 4.3. Realizing the difficulties of achieving continuous delivery, organizations have not considered further steps.

It could be that end-to-end CI/CD as presented in our model, see Fig 2, and specific steps following continuous deployment are not feasible for products with a high level of customer customization, strict service level agreements, and a general push-back from existing customers. Furthermore, it could be that end-to-end CI/CD requires a special context granting the software vendor control over the pipeline and access to customer data.

Fitzgerald et al. (Fitzgerald and Stol 2017) argue that agile software development methodologies, such as scrum, are finding their way into regulated domains. Nevertheless, literature offers little support in retrofitting existing organizations with CI/CD capabilities. In particular, organizational interfaces such as business models, software delivery models, customer agreements are assumed to be flexible and adjustable without cost.

As shown by our two cases, organizations can benefit from automating and streamlining the internal development process without considering the complete end-to-end CI/CD. Thus, the focus on adopting CI/CD in complex domains could be to maximize the benefits from internal automation steps first, see Table 9.

Importantly, our empirical findings show that companies acknowledge the potential benefits end-to-end CI/CD pipeline as presented in Fig. 2, and Tables 4–7. However, when retrofitting existing offerings with CI/CD, organizations are cautious to make sweeping changes and to take risks that can disrupt their business. For instance, requiring customers to accept continuous deliveries and share data may backfire if customers are not prepared and ready to comply. Thus, the investments (costs) and associated risks nullify the potential benefits to adopting the practices, see Section 4.1.7.

5.2 What Costs and Benefits Steer the Adoption of the CI/CD in Practice?

In Ericsson's case, the primary driver to adopt CI/CD is to streamline and speed up internal software delivery processes. The main challenge to solve is integrating a large number of software components into customized customer solutions efficiently, see Section 4.1.4.

The secondary objective is to improve their business models based on CI/CD capabilities. However, what exactly are these capabilities, the extent to which they are relevant to current customers, and how to retrofit current offerings to be compatible with the new business models remains unknown. As a consequence, the organization currently focuses on internal benefits, see Section 4.1.6.

In Telia, the only driver for adopting the CI/CD practices is to improve the internal processes and free up resources for more value-adding activities, see Section 4.2.4. While the organization realizes the potential benefits of continuous software deliveries, attaining them is not their current agenda.

During our workshops, the participants reflected that engineers generally welcome test and build automation initiatives, containerization, and other technical practices. The value of introducing such practices is immediate and substantial. However, the challenge is to convince management to invest in other CI/CD practices spanning multiple organizational units and organizational functions especially, if that requires adjustments in currently operational business models. For instance, pushing customer to accept more frequent deliveries and share data, see Tables 4–7.

Participants expressed that they realize the potential of developing new collaborative business models based on CI/CD capabilities. However, due to their organizational complexities and potential customer push-back, changing their business models on a scale is beyond their current planning. More analysis is required to understand how changes in business models could help to attain the organizational objectives and what are the constraints of such changes, see the discussion on Sections 4.1.6 and 4.3.

5.3 What is Needed to Support the CI/CD Cost-Benefit Analysis?

In both studied cases, we observed a difficulty to extend the continuous pipeline outside the development organization to customers and end-users. The software delivery process is an important feature and changing it changes the total value of the offering (Khurum et al. 2013). The two cases demonstrate that predictability and risk minimization in software deliveries takes precedence over delivery speed in domains where software is business-critical, see Sections 4.1–4.2.

The literature emphasizes that delivery speed and frequency is a solution to nearly all software delivery challenges, see for instance Humble et al. (Humble and Kim 2018). However, there is little discussion on weighing the benefits of schedule-based software releases versus faster access to new features from customers' perspectives. For example, in customers' eyes, the fact that software does not change could be an important feature if a software upgrade requires updating many intricate dependencies.

Avenue for further work: There is an opportunity to explore the mapping between software value aspects (Khurum et al. 2013) and benefits and trade-offs of continuous engineering. That is, knowing what value aspects are prioritized by customers and the software vendor enables a more fine-grained analysis of how continuous engineering contributes to these aspects.

We observe that organizations are often not aware of what specific goals they aim to attain by adopting CI/CD, becoming better in delivering software and what are the constraints of any solutions they wish to implement. Attaining one goal, e.g. increasing speed by implementing automation and eliminating manual steps, could imply more standardization. However, that can be counterproductive if customers expect individual treatment and bespoke solutions. This issue is evident in Case I, see Section 4.1.6, where the organization contemplates the benefits of new collaborative business models, however there is not enough details to gauge the relevance of such models in their domain.

Avenue for further work: There is an opportunity to devise a methodology to identify and break down organizational goals towards software delivery and estimate the degree of freedom to implement any changes. The adoption of any new practices or tools should be guided by specific and measurable goals. Importantly, the software delivery pipeline transcends individual organizational departments and extends into a customer organization. Therefore, the goals, KPIs, and objectives should aligned and shared across the organization and accepted by the customers.

More understanding of the organizational benefits is needed to support management decisions concerning adopting CI/CD and driving the necessary organizational changes. Both engineering and organizational literature, see Section 4.4 and, e.g. Hanelt et al. (Hanelt et al. 2021), discusses organizational benefits such as cooperative business models and telemetry to tailor the offering to specific customer needs. However, it is unknown how to gauge relevance, quantify, and compare these benefits to the required investments.

Avenue for further work: The relationship between digital transformation and continuous engineering in software organizations need to be explored. Digital transformation explores the organizational benefits of adopting technology to streamline their processes. Continuous engineering focus on the benefits from streamlining software engineering. Exploring how one could enable and support another could help establishing a joint management-engineering view on the associated changes, benefits and required investments.

We further notice that the boundaries of control constrain the adoption of CI/CD. In the studied cases, the software vendors only partially control the delivery pipeline. By design, the final installation and release to end-users is out of control of the software vendor, see Section 4.1.3. The literature overlooks this and assumes that the software vendor has complete control over the pipeline. Expanding control boundaries and assuming more responsibilities requires substantial changes in the offering and the underlying business model. It also implies reducing the level of control other stakeholders has over the software. More understanding is needed to gauge the implications of extending boundaries of control.

Avenue for further work: More research is required how to integrate different pipelines of software delivery with potentially different aims, release cadences, stakeholders, and controlled by different organizations. Moreover, large organizations, such as Ericsson, may have multiple internal pipelines that could branch and merge in different ways and are controlled by different internal stakeholders. It remains to be explored how to streamline and synchronize software delivery in such scenarios.

6 Conclusions

This paper has examined the adoption of continuous software engineering practices in two industrial cases and performed a literature study to understand the costs and benefits perspective on the CI/CD.

We found that literature overstates the benefits of CI/CD without recognizing specific domain complexities and the challenges of retrofitting already existing offerings with CI/CD capabilities. Importantly, literature overlooks the customer perspective on accepting continuous software deliveries and provides little guidance on upgrading existing customers to continuous mode.

The two case studies reveal that the adoption of CI/CD in organizations starts from the bottom-up engineering teams attempting to automate and streamline their work. However, expanding the continuous software delivery pipeline to adjacent organizational units, especially to customer organizations, is challenging. A significant challenge is to gauge the relevance of CI/CD benefits for a specific organizational domain and context.

We identify a need to explore customer further and organizational perspectives and understand the contextual requirements for adopting continuous software engineering practices. Furthermore, we identify a gap in state-of-the-art concerning retrofitting an existing product with a CI/CD pipeline.

Acknowledgements The authors would like to thank all study participants for their time dedicated to this study. We would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

Funding Open access funding provided by Blekinge Institute of Technology.

References

Alahyari H., Gorschek T., Svensson R. B. (2019) An exploratory study of waste in software development organizations using agile or lean approaches: A multiple case study at 14 organizations. Information and Software Technology 105:78–94

Alahyari H., Svensson R. B., Gorschek T. (2017) A study of value in agile software development organizations. Journal of Systems and Software 125:271–288

Anderson J., Azizi M., Salem S., Do H. (2019) On the use of usage patterns from telemetry data for test case prioritization. Information and Software Technology 113:110–130

Balalaie A., Heydarnoori A., Jamshidi P. (2016) Microservices architecture enables devops: Migration to a cloud-native architecture. Ieee Software 33(3):42–52

Boehm B. (2003) Value-based software engineering. ACM SIGSOFT Software Engineering Notes 28(2):4

Bosch J. (2012) Building products as innovation experiment systems. In: International Conference of Software Business, Springer, pp 27–39

Carriere J., Kazman R., Ozkaya I. (2010) A cost-benefit framework for making architectural decisions in a business context. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, IEEE. vol. 2, pp 149–157

Chandra D. G., Borah M. D. (2012) Cost benefit analysis of cloud computing in education. In: 2012 International Conference on Computing, Communication and Applications, IEEE, pp 1–6

Chen L. (2015) Continuous delivery: Huge benefits, but challenges too. IEEE Softw 32(2):50–54

Chen L. (2017) Continuous delivery: Overcoming adoption challenges. J Syst Softw 128:72–86

Chen L. (2018) Microservices: architecting for continuous delivery and devops. In: 2018 IEEE International conference on software architecture (ICSA), IEEE, pp 39–397

Claps G. G., Svensson R. B., Aurum A. (2015) On the journey to continuous deployment: Technical and social challenges along the way. Information and Software Technology 57:21–31

Cole R. E. (2001) From continuous improvement to continuous innovation. Qual Manag J 8(4):7–21

Dännart S., Constante F. M., Beckers K. (2019) An assessment model for continuous security compliance in large scale agile environments. In: International Conference on Advanced Information Systems Engineering, Springer, pp 529–544

Del Rosso C. (2006) Continuous evolution through software architecture evaluation: a case study. Journal of Software Maintenance and Evolution: Research and Practice 18(5):351–383

Drèze J., Stern N. (1987) The theory of cost-benefit analysis. In: Handbook of public economics, Elsevier. vol 2, pp 909–989

Dubey A., Wagle D. (2007) Delivering software as a service. McKinsey Q 6(2007):2007

Ehlers J., van Hoorn A., Waller J., Hasselbring W. (2011) Self-adaptive software system monitoring for performance anomaly localization. In: Proceedings of the 8th ACM international conference on Autonomic computing, pp 197–200

Elazhary O., Werner C., Li Z. S., Lowlind D., Ernst N. A., Storey M.-A. (2021) Uncovering the benefits and challenges of continuous integration practices. IEEE Trans Softw Eng

Fabijan A., Dmitriev P., Olsson H. H., Bosch J. (2017) The evolution of continuous experimentation in software product development: from data to a data-driven organization at scale. In: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), IEEE, pp 770–780

Fagerholm F., Guinea A. S., Mäenpää H., Münch J. (2017) The right model for continuous experimentation. J Syst Softw 123:292–305

Fatima A., Nazir N., Khan M. G. (2017) Data cleaning in data warehouse: A survey of data pre-processing techniques and tools. IJ Information Technology and Computer Science 3:50–61

Feitelson D. G., Frachtenberg E., Beck K. L. (2013) Development and deployment at facebook. IEEE Internet Computing 17(4):8–17

Felidré W., Furtado L., da Costa D. A., Cartaxo B., Pinto G. (2019) Continuous integration theater. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, pp 1–10

Fitzgerald B., Stol K.-J. (2014) Continuous software engineering and beyond: trends and challenges. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pp 1–9

Fitzgerald B., Stol K.-J. (2017) Continuous software engineering: A roadmap and agenda. J Syst Softw 123:176–189

Fitzgerald B., Stol K.-J., O'Sullivan R., O'Brien D. (2013) Scaling agile methods to regulated environments: An industry case study. In: 2013 35th International Conference on Software Engineering (ICSE), IEEE, pp 863–872

Fowler M., Highsmith J. et al (2001) The agile manifesto. Software Development 9(8):28–35

Frank M. (2000) Engineering systems thinking and systems thinking. Syst Eng 3(3):163–168

Frow N., Marginson D., Ogden S. (2010) "continuous"? budgeting: Reconciling budget flexibility with budgetary control. Acc Organ Soc 35(4):444–461

Gardner S. R. (1998) Building the data warehouse. Commun ACM 41(9):52–60

Garousi V., Felderer M. (2016) Developing, verifying, and maintaining high-quality automated test scripts. IEEE Softw 33(3):68–75

Gefen D., Karahanna E., Straub D. W. (2003) Trust and tam in online shopping: An integrated model, MIS quarterly pp 51–90

Genc-Nayebi N., Abran A. (2017) A systematic literature review: Opinion mining studies from mobile app store user reviews. J Syst Softw 125:207–219

Giardino C., Paternoster N., Unterkalmsteiner M., Gorschek T., Abrahamsson P. (2015) Software development in startup companies: the greenfield startup model. IEEE Trans Softw Eng 42(6):585–604

Gustafsson A., Johnson M. D., Roos I. (2005) The effects of customer satisfaction, relationship commitment dimensions, and triggers on customer retention. Journal of marketing 69(4):210–218

Guzman E., Ibrahim M., Glinz M. (2017) Prioritizing user feedback from twitter: A survey report. In: 2017 IEEE/ACM 4th International Workshop on CrowdSourcing in Software Engineering (CSI-SE), IEEE, pp 21–24

Hanelt A., Bohnsack R., Marz D., Antunes Marante C. (2021) A systematic review of the literature on digital transformation: insights and implications for strategy and organizational change. J Manag Stud 58(5):1159–1197

Hasselbring W., Steinacker G. (2017) Microservice architectures for scalability, agility and reliability in e-commerce. In: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), IEEE, pp 243–246

Hazzan O., Dubinsky Y. (2009) Agile software engineering. Springer Science & Business Media

Hilton M., Nelson N., Tunnell T., Marinov D., Dig D. (2017) Trade-offs in continuous integration: assurance, security, and flexibility. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp 197–207

Hilton M., Tunnell T., Huang K., Marinov D., Dig D. (2016) Usage, costs, and benefits of continuous integration in open-source projects. In: 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 426–437

Huang X., Zhang H., Zhou X., Babar M. A., Yang S. (2018) Synthesizing qualitative research in software engineering: A critical review. In: Proceedings of the 40th International Conference on Software Engineering, pp 1207–1218

Humble J., Kim G. (2018) Accelerate: The science of lean software and devops: Building and scaling high performing technology organizations. IT Revolution

Isaak J., Hanna M. J. (2018) User data privacy: Facebook, cambridge analytica, and privacy protection. Computer 51(8):56–59

Jalali S., Wohlin C. (2012) Global software engineering and agile practices: a systematic review. Journal of software: Evolution and Process 24(6):643–659

Jiang J., Yang Y., He J., Blanc X., Zhang L. (2017) Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. Inf Softw Technol 84:48–62

Johnson P. M., Kou H., Paulding M., Zhang Q., Kagawa A., Yamashita T. (2005) Improving software development management through software project telemetry. IEEE software 22(4):76–85

Kasurinen J., Taipale O., Smolander K. (2010) Software test automation in practice: empirical observations. Advances in Software Engineering (2010)

Khurum M., Gorschek T., Wilson M. (2013) The software value map—an exhaustive collection of value aspects for the development of software intensive products. Journal of software: Evolution and Process 25(7):711–741

Kim S., Park S., Yun J., Lee Y. (2008) Automated continuous integration of component-based software: An industrial experience. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, IEEE, pp 423–426

King J. L., Schrems E. L. (1978) Cost-benefit analysis in information systems development and operation. ACM Computing Surveys (CSUR) 10(1):19–34

Kitchenham B. (2004) Procedures for performing systematic reviews. Keele, UK, Keele University 33(2004):1–26

Kumar D., Mishra K. K. (2016) The impacts of test automation on software's cost, quality and time to market. Procedia Computer Science 79:8–15

Kuula S., Haapasalo H. (2017) Continuous and co-creative business model creation. In: Service business model innovation in healthcare and hospital management, Springer, pp 249–268

Lacoste F. J. (2009) Killing the gatekeeper: Introducing a continuous integration system. In: 2009 agile conference, IEEE, pp 387–392

Lam W., Godefroid P., Nath S., Santhiar A., Thummalapenta S. (2019) Root causing flaky tests in a large-scale industrial setting. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 101–111

Laukkanen E., Itkonen J., Lassenius C. (2017) Problems, causes and solutions when adopting continuous delivery-a systematic literature review. Inf Softw Technol 82:55–79

Laukkanen E., Paasivaara M., Arvonen T. (2015) Stakeholder perceptions of the adoption of continuous integration–a case study. In: 2015 agile conference, IEEE, pp 11–20

Lehtola L., Kauppinen M., Vähäniitty J., Komssi M. (2009) Linking business and requirements engineering: is solution planning a missing activity in software product companies? Requirements engineering 14(2):113–128

Letier E., Stefan D., Barr E. T. (2014) Uncertainty, risk, and information value in software requirements and architecture. In: Proceedings of the 36th International Conference on Software Engineering, pp 883–894

Li Y., Chang K.-C., Chen H.-G., Jiang J. J. (2010) Software development team flexibility antecedents. J Syst Softw 83(10):1726–1734

Lin C. (2018) Data driven product management. IEEE Eng Manag Rev 46(1):16–18

Loebbecke C., Picot A. (2015) Reflections on societal and business model transformation arising from digitization and big data analytics: A research agenda. The Journal of Strategic Information Systems 24(3):149–157

Lohan G. (2013) A brief history of budgeting: reflections on beyond budgeting, its link to performance management and its appropriateness for software development. In: International Conference on Lean Enterprise Software and Systems, Springer, pp 81–105

Mäkinen S., Leppänen M., Kilamo T., Mattila A.-L., Laukkanen E., Pagels M., Männistö T. (2016) Improving the delivery cycle: A multiple-case study of the toolchains in finnish software intensive enterprises. Inf Softw Technol 80:175–194

Maresova P., Sobeslav V., Krejcar O. (2017) Cost–benefit analysis–evaluation model of cloud computing deployment for use in companies. Appl Econ 49(6):521–533

Martin R. C. (2002) Agile software development: principles, patterns, and practices. Prentice Hall

McHugh M., McCaffery F., Fitzgerald B., Stol K.-J., Casey V., Coady G. (2013) Balancing agility and discipline in a medical device software organisation. In: International Conference on Software Process Improvement and Capability Determination, Springer, pp 199–210

Memon A., Gao Z., Nguyen B., Dhanda S., Nickell E., Siemborski R., Micco J. (2017) Taming google-scale continuous testing. In: 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), IEEE, pp 233–242

Meyer M. (2014) Continuous integration and its tools. IEEE software 31(3):14–16

Moyon F., Beckers K., Klepper S., Lachberger P., Bruegge B. (2018) Towards continuous security compliance in agile software development at scale. In: 2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE), IEEE, pp 31–34

Neely S., Stolt S. (2013) Continuous delivery? easy! just change everything (well, maybe it is not that easy). In: 2013 Agile Conference, IEEE, pp 121–128

O'Connor R. V., Elger P., Clarke P. M. (2017) Continuous software engineering-a microservices architecture perspective. Journal of Software: Evolution and Process 29(11):e1866

Olsson H. H., Alahyari H., Bosch J. (2012) 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, IEEE, pp 392–399

Olsson T., Wnuk K. (2018) Qreme–quality requirements management model for supporting decision-making. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, pp 173–188

Ouriques R. A. B., Wnuk K., Gorschek T., Svensson R. B. (2019) Knowledge management strategies and processes in agile software development: a systematic literature review. International journal of software engineering and knowledge engineering 29(03):345–380

Petersen K., Feldt R., Mujtaba S., Mattsson M. (2008) Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12, pp 1–10

Pinto G., Castor F., Bonifacio R., Rebouças M. (2018) Work practices and challenges in continuous integration: A survey with travis ci users. Software: Practice and Experience 48(12):2223–2236

Poppendieck M. et al (2011) Principles of lean thinking. IT Management Select 18(2011):1–7

Poppendieck M., Poppendieck T. (2003) Lean software development: An agile toolkit: An agile toolkit. Addison-Wesley

Provost F., Fawcett T. (2013) Data science and its relationship to big data and data-driven decision making. Big data 1(1):51–59

Rahm E., Do H. H. (2000) Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23(4):3–13

Raulamo-Jurvanen P., Mäntylä M., Garousi V. (2017) Choosing the right test automation tool: a grey literature review of practitioner sources. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, pp 21–30

Riaz M., Sulayman M., Naqvi H. (2009) Architectural decay during continuous software evolution and impact of 'design for change'on software architecture. In: International Conference on Advanced Software Engineering and Its Applications, Springer, pp 119–126

Rodríguez P., Haghighatkhah A., Lwakatare L. E., Teppola S., Suomalainen T., Eskeli J., Karvonen T., Kuvaja P., Verner J. M., Oivo M. (2017) Continuous deployment of software intensive products and services: A systematic mapping study. J Syst Softw 123:263–291

Rodríguez P., Markkula J., Oivo M., Garbajosa J. (2012) Analyzing the drivers of the combination of lean and agile in software development companies. In: International Conference on Product Focused Software Process Improvement, Springer, pp 145–159

Rogers R. O. (2004) Scaling continuous integration. In: International conference on extreme programming and agile processes in software engineering, Springer, pp 68–76

Romano Jr N. C., Pick J. B., Roztocki N. (2010) A motivational model for technology-supported cross-organizational and cross-border collaboration. Eur J Inf Syst 19(2):117–133

Runeson P., Host M., Rainer A., Regnell B. (2012) Case study research in software engineering: Guidelines and examples. John Wiley & Sons

Ryals L. (2005) Making customer relationship management work: the measurement and profitable management of customer relationships. Journal of marketing 69(4):252–261

Sassone P. G., Schaffer W. A. (1978) Cost-benefit analysis: a handbook, vol 182. Academic Press, New York

Senapathi M., Buchan J., Osman H. (2018) 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, pp 57–67

Serrat O. (2017) Bridging organizational silos. In: Knowledge Solutions. Springer, pp 711–716

Shahin M., Babar M. A., Zahedi M., Zhu L. (2017) Beyond continuous delivery: an empirical investigation of continuous deployment challenges. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, pp 111–120

Shahin M., Babar M. A., Zhu L. (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access 5:3909–3943

Shahin M., Zahedi M., Babar M. A., Zhu L. (2019) An empirical study of architecting for continuous delivery and deployment. Empir Softw Eng 24(3):1061–1108

Shamshiri S., Just R., Rojas J. M., Fraser G., McMinn P., Arcuri A. (2015) Do automatically generated unit tests find real faults? an empirical study of effectiveness and challenges (t). In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 201–211

Sidky A., Arthur J., Bohner S. (2007) A disciplined approach to adopting agile practices: the agile adoption framework. Innovations in systems and software engineering 3(3):203–216

Stolberg S. (2009) Enabling agile testing through continuous integration. In: 2009 agile conference, IEEE, pp 369–374

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

Sundelin A., Gonzalez-Huerta J., Wnuk K. (2018) Test-driving fintech product development: An experience report. In: International Conference on Product-Focused Software Process Improvement, Springer, pp 219–226

Sundelin A., Gonzalez-Huerta J., Wnuk K. (2020) The hidden cost of backward compatibility: when deprecation turns into technical debt-an experience report. In: Proceedings of the 3rd International Conference on Technical Debt, pp 67–76

Susarla A., Barua A., Whinston A. B. (2009) A transaction cost perspective of the" software as a service" business model. J Manag Inf Syst 26(2):205–240

Tómasdóttir K. F., Aniche M., van Deursen A. (2017) Why and how javascript developers use linters. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 578–589

van Hoorn A., Rohr M., Hasselbring W., Waller J., Ehlers J., Frey S., Kieselhorst D. (2009) Continuous monitoring of software services: Design and application of the kieker framework

Vasilescu B., Yu Y., Wang H., Devanbu P., Filkov V. (2015) Quality and productivity outcomes relating to continuous integration in github. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp 805–816

Wiklund K., Eldh S., Sundmark D., Lundqvist K. (2017) Impediments for software test automation: A systematic literature review. Software Testing, Verification and Reliability 27(8):e1639

Williams L., Kudrjavets G., Nagappan N. (2009) On the effectiveness of unit test automation at microsoft. In: 2009 20th International Symposium on Software Reliability Engineering, IEEE, pp 81–89

Williams L., Maximilien E. M., Vouk M. (2003) Test-driven development as a defect-reduction practice. In: 14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003., IEEE, pp 34–45

Yaman S. G., Sauvola T., Riungu-Kalliosaari L., Hokkanen L., Kuvaja P., Oivo M., Männistö T. (2016) Customer involvement in continuous deployment: a systematic literature review. In: International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, pp 249–265

Yu L., Alégroth E., Chatzipetrou P., Gorschek T. (2020) Utilising ci environment for efficient and effective testing of nfrs. Inf Softw Technol 117:106199

Zhang D. (2018) Big data security and privacy protection. In: 8th International Conference on Management and Computer Science (ICMCS 2018), Atlantis Press, pp 275–278

Zhu L., Bass L., Champlin-Scharff G. (2016) Devops and its practices. IEEE Softw 33(3):32–34

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.