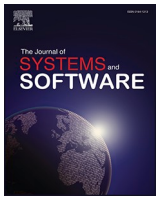
The Journal of Systems and Software 213 (2024) 112056

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



The never-ending story–How companies transition to and sustain continuous software engineering practices

Jacob Nørbjerg [a,*], Yvonne Dittrich [b]

[a] Department of Digitalization, Copenhagen Business School, 60 Howitzvej, Frederiksberg 2000, Denmark
[b] Computer Science Department, IT University of Copenhagen, Rued Langgaardsvej 7, Copenhagen S, 2300, Denmark

ARTICLE INFO

Keywords:
Continuous software engineering
Infrastructuring
Software practice
Software architecture
Empirical study

STRUCTURED ABSTRACT

Context: – There is increasing interest in Continuous Software Engineering (CSE) among practitioners and researchers. CSE addresses the need to increase flexibility and short release cycles, especially when augmenting software as a service, without jeopardizing software quality.
Purpose/objectives: – Empirical literature focuses on the transition to CSE as introducing a new method supported by new tools and architectural concepts. Little is known, however, about how software companies sustain CSE practices.
Design/methodology/approach: – The analysis proceeds in two stages. First, we present a thematic analysis based on qualitative interviews with both management and developers from three different software development organizations. Then we apply the concept of infrastructuring to the results as a sense-making device.
Findings: - We show how companies adapt and align their CSE organization, processes, and techniques to internal and external demands and conditions, resulting in widely varying practices. We further see that CSE in the companies is unlikely to arrive at a stable state. Rather, the companies continuously adapt their practices due to changes in the environment, requirements, new techniques and tools, and new software dependencies.
Implications/value: – We use the concept of 'infrastructuring' from computer supported cooperative work and information systems to make sense of the continuous change we see in our interviews. We show that CSE needs to be regarded as a new way to make use of method (elements), processes and tools in software engineering, rather than a new method.

1. Introduction

Many software development organizations turn their attention to Continuous Software Engineering (CSE) (Fitzgerald and Stol, 2015; O'Connor et al., 2017; Shahin et al., 2018). CSE introduces a continuous – daily or hourly – flow of incremental software releases (Dennehy and Conboy, 2016; Fitzgerald and Stol, 2015). Thus, CSE answers the need for increased flexibility, shortened release cycles, higher quality, and better productivity that has driven the agile software development movement since its inception in the early 2000s (Beck et al., 2001; Conboy, 2009; Dennehy and Conboy, 2016). Even the current two-week sprints (down from 5 to 6 weeks in the early 2000s) are too long to satisfy the modern organization's need for rapid adaptation of its organizational IT systems to changing business needs (Agarwal and Tiwana, 2015; Dennehy and Conboy, 2016; Klotins and Peretz-Andersson, 2022).

CSE originated in large online service providers such as Microsoft, Google, Amazon, and Facebook (Guckenheimer, 2015; Metz, 2016; Savor et al., 2016). These companies developed tools and processes to correct errors and update functionality in on-line applications without downtime, and they replaced scheduled releases with continuous development and automated build, test, and deployment. Companies in other sectors, e.g., financial institutions, telecommunications, and embedded systems, also explore how CSE can enable rapid responses to changing customer demands and new business opportunities (cf., Agarwal and Tiwana, 2015; Klotins and Peretz-Andersson, 2022; Lwakatare et al., 2016).

The transition to CSE is complex and changes a company's software development and delivery technologies and processes, i.e., software architecture and design, software development, integration and deployment technologies, software development management, and customer relationships (cf., Hemon et al., 2020; Klotins et al., 2022;

* Corresponding author.
E-mail address: jno.digi@cbs.dk (J. Nørbjerg).

https://doi.org/10.1016/j.jss.2024.112056
Received 13 March 2023; Received in revised form 20 March 2024; Accepted 4 April 2024
Available online 5 April 2024

Laukkanen et al., 2017).

Existing CSE research focuses on the selection and implementation of software architectures and technologies for integration and deployment, but there is not much research into how CSE affects software development practices and organizations (Elazhary et al., 2022; Klotins et al., 2022). There is furthermore a need of research into how organizations sustain and continuously improve CSE technologies, processes, and the supporting organizational structures (Fitzgerald and Stol, 2015; Klotins et al., 2022).

In this paper we ask the following research question: *How do software organizations initiate and sustain CSE practices, processes, and technologies?* The paper is based on a study of the implementation and evolution of CSE in three companies. Our initial inductive analysis of the CSE practices and technologies in the case organizations revealed, first, that organizations take different paths towards the implementation of CSE. Second, we found that development and deployment practices and routines, as well as development and deployment technologies, were not stable but changed over time. We furthermore observed that changes were interdependent and originated from many sources, e.g., when unexpected dependencies in the product architecture affected inter-team coordination practices, or when developers adapted their practices to new development platforms. We finally observed that the CSE transition did not result in a uniform set of practices, processes, organizational structures, and technologies. Instead, old, and new software architectures, development and deployment technologies, and associated practices co-exist, mainly due to the cost and effort required to build and maintain a consistent set of practices and technologies in a changing world.

We turned to the (information) *infrastructures* and *infrastructuring* literature (cf., Henfridsson and Bygstad, 2013; Star and Ruhleder, 1996) for a conceptual framework to make sense of the findings. These concepts, which have been applied to the study of organizational as well as interorganizational information systems (Giraldo-Mora, 2023; Henfridsson and Bygstad, 2013) offer "powerful lenses for conceptualizing the increasingly interconnected information system collectives found in contemporary organization" (Henfridsson and Bygstad, 2013, p. 907). They have also been used to study the configuration of practices, norms, technologies and structures in relation to software work (cf., Karasti and Blomberg, 2018; Pipek and Wulf, 2009). A central tenet of this research is that infrastructures are constantly negotiated and emerging due to internal and external factors, changing priorities, and technologies.

Based on our results, we argue, first, that each organization implements CSE in its own way depending on internal as well as external conditions and circumstances. Second, that CSE will not evolve towards a stable or final state. The processes, techniques and tools discussed under the CSE heading comprise an evolving toolbox of technological and methodological components. Companies and teams must continuously readjust and combine the way they use these components to meet changing needs and exploit new opportunities. We finally argue that the ongoing adoption and adaptation of CSE practices and technologies is a separate and important new task that requires attention and resources and, perhaps, the formation of new organizational structures (cf., Fitzgerald and Stol, 2015; Leite et al., 2021). More research into the dynamic and evolving nature of CSE is needed to further expand and verify these findings (Fitzgerald and Stol, 2015).

We make the following contributions to the socio-technical research on CSE: We show the different paths companies can take towards CSE, and that the CSE infrastructure is never perfect or homogenous but evolves and adapts to meet changing, and sometimes conflicting internal and external needs and conditions. We further show how the concepts of infrastructure and infrastructuring, that we adapt from discussions in Information Systems, Human computer Interaction and Computer Supported Cooperative Work, can be used to understand the continuous adaptation of software development tools, methods, and processes as part of everyday software development that we see in the interviews.

The remainder of this paper is structured as follows: the next section introduces related work on CSE and the theoretical background on infrastructures and infrastructuring. We then present our research methods and discuss the trustworthiness and limitations of the research. Section 4 presents the analysis of the cases. Section 5 analyses the findings from the cases through the lens of the infrastructure and infrastructuring. In Section 6 we discuss the findings. Section 7 concludes the article.

2. Background and related work

Continuous Software Engineering (CSE) is an umbrella term for software production practices and technologies that enable software development organizations to release new versions of their software product(s) in small increments several times per day (Fitzgerald and Stol, 2015; Klotins et al., 2022; Shahin et al., 2017). CSE replaces the timeboxed planning, development, and delivery of software in agile development with a continuous flow of small increments to working software (Dennehy and Conboy, 2016).

CSE relies on an automated or partly automated pipeline from code commit to operations (Klotins et al., 2022). The pipeline is commonly described as consisting of the following elements: continuous integration (CI), continuous delivery (CDE), and continuous deployment (CD) (see Fig. 1). CI enables software development organizations to integrate new code into the codebase hourly or daily, and it is a prerequisite for the other components in the pipeline. The next step, CDE, prepares the software for deployment, and CD closes the gap between development and deployment by automatically pushing the new version of the software to production after the automated acceptance test. Note the feedback to developers from all stages in the process.

There is a close relationship between CSE and the rapid evolution of modern businesses (Fitzgerald and Stol, 2015; Osmundsen and Bygstad, 2021). Fitzgerald and Stol (2015) uses the term continuous* to emphasize how CSE connects to the business in general. Continuous* covers the cycle from business planning and budgeting, over software development and operations, to customer feedback and continuous innovation and improvement of the software product and development practices. This paper focuses on the software development and deployment parts of continuous* – including planning, management and improvement. We use CSE as a shorthand for these activities.

2.1. Introducing CSE

Organizations often move towards CSE from an agile process supported by CI, CDE and CD technologies (Hemon et al., 2020; Shahin et al., 2017). The transition is complex and involves changes to the software product architecture, software development practices and delivery technologies, as well as to the organization, developer skills, and development management. (Hemon et al., 2020; Leite et al., 2021; O'Connor et al., 2017; Shahin et al., 2017). The monolithic architecture of legacy systems, including those built in an agile manner, is unsuited for several daily builds and deployments (Laukkanen et al., 2017) and a comprehensive, lengthy and costly refactoring of the software product into smaller components, e.g., microservices, is required to meet the needs of CSE (Laukkanen et al., 2017; Leppänen et al., 2015; Shahin et al., 2018)

CSE cannot be reduced to a single set of practices and technologies (Klotins et al., 2022) and finding the approach to CSE most suited for the organization is complicated (Shahin et al., 2018). Bellomo (2014) describes 15 different architectural and design tactics that companies must consider for CSE. The tactics range from a complete architectural overhaul, such as moving to microservices, to parametrization of database and server names. The tactics are interrelated, which further complicates the CSE implementation. Claps (2015) identifies 9 technical transition challenges and mitigation strategies, and Shahin et al. (2017) identified over 30 approaches to the technical transition to CSE.

The costs of fully implementing the complete continuous* cycle

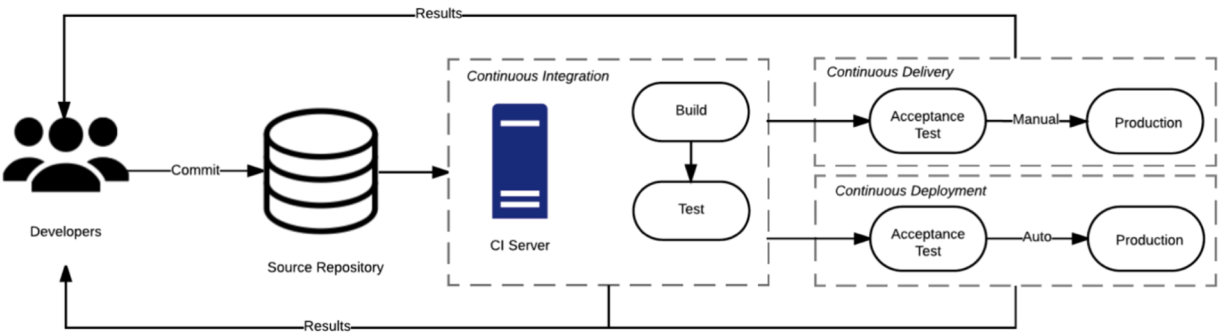


Fig. 1. The relationship between continuous integration, delivery, and deployment as presented by Shahin et al. (2017, p. 2).

(Fitzgerald and Stol, 2015) may outweigh the benefits (Elazhary et al., 2022; Klotins et al., 2022; Shahin et al., 2018). Klotins et al. (2022) recommends a systematic and planned transition guided by careful cost-benefit analyses. Elazhary et al. (2022) note that CSE practices and solutions can create new challenges, e.g., a microservices architecture can result in complex interdependency issues (cf., Laukkanen et al., 2017; O'Connor et al., 2017).

2.2. Organizational change

The technical transition is intertwined with changes to the software development organization and software development skills and practices but there are few studies of the social and organizational side of CSE implementation (Elazhary et al., 2022; Klotins and Peretz-Andersson, 2022; Leite et al., 2021).

Leite et al. (2021) identify four organizational structures along the path from agile development to CI/CD: (1) siloed departments, (2) classical DevOps, (3) cross-functional teams, and (4) platform teams. The platform team – or build sheriff (Claps et al., 2015; Shahin et al., 2017) – is a separate organizational entity responsible for the delivery pipeline and the practices and norms for coordinating development and deployment.

Individual developers experience increased pressure from the steady flow of tasks in CSE, and may not be prepared for the responsibility for deployment decisions (Claps et al., 2015; Hemon et al., 2020; Neely and Stolt, 2013).

As far as we know, only few papers theorize the relationship between CSE technologies and the social and organizational context (Elazhary et al., 2021; Fitzgerald and Stol, 2015). Elazhary (2021) introduces a socio-technical model of CI with four interacting elements: automation, process, documentation, and team member. They demonstrate how the model may *explain* a specific CI configuration of the four elements in CI, but they do not discuss the dynamics of how and why a configuration evolves over time. Their study is also limited to the CI activities.

2.3. Variations in implementation

The majority of CSE research focuses on the challenges and solutions in CI/CDE/CD implementation, but there is little research into the other activities in the complete continuous* cycle (Klotins et al., 2022; Elazhary, 2022).

The automated CI/CD pipeline results in immediate and tangible benefits in most cases but other activities in the continuous* cycle such as systematic customer feedback, or continuous planning and improvement are harder to implement, and their benefits have not been documented by research (Klotins and Peretz-Andersson, 2022).

Furthermore, the core CSE pipeline from development to deployment may be incomplete due to contextual factors or changing priorities (Leite et al., 2021). Leppänen (2015) observes, for example, that refactoring legacy code may be incomplete due to resource constraints or changed priorities, leaving pockets of old program code in the product. The result is an incoherent architecture and different development and deployment technologies and processes to support the different parts of the software.

CSE adoption is an emerging process that adapts to new technologies and other changes (Elazhary et al., 2021; O'Connor et al., 2017). Thus, we cannot describe or understand CSE as a particular set of organizational structures, practices, and technologies. Each organization adopts and adapts organizational structures, practices and technologies to fit its needs and other contextual factors (Elazhary et al., 2021; Klotins and Gorschek, 2022; Leite et al., 2021). It also follows that we cannot decide when a CSE transition is complete, i.e., what constitutes the minimal configuration of practices and technologies in CSE.

To the best of our knowledge, however, there is not much research into *how* organizations implement and continuously adapt CSE (Elazhary et al., 2021; Klotins and Gorschek, 2022). O'Connor (2017) applies a model with 44 situational factors that influence the software process to a case study of CSE. The study is, however, limited to the role of changing technologies in shaping the CSE process. "[T]he process itself is subject to change because of technology strengths and limitations (p. 8)." In this paper we use infrastructure and infrastructuring (see the following section) to show how practices, technologies and organizational structures interact in the continuous shaping of CSE from development to deployment.

2.4. Infrastructures and infrastructuring

We draw on the concepts of infrastructures and infrastructuring to better understand the variation and the continuity of change in the implementation of CSE that we saw in our interviews,: Our understanding of infrastructure is based in the research traditions Information Systems, Computer Supported Cooperative Work (Karasti and Blomberg, 2018) and Society and Technology Studies (Karasti and Blomberg, 2018; Pipek et al., 2017; Henfridsson and Bygstad, 2013; Pipek and Wulf, 2009; Star and Ruhleder, 1996). Here, infrastructures and infrastructuring are used to understand the relationship between the technical support for a collaborative activity and the social protocols that shape how the technical installations become supportive for certain practices. In this literature, an infrastructure consists of the material support for societal, organizational, or merely collaborative activities, e.g., roads, or networked computers, *as well as* the relations between material or informational structures on the one hand, and the rules, norms, and (human) practices the material infrastructures support on the other (Star and Ruhleder, 1996). According to this view, for example, the traffic rules that regulate how to use roads and sidewalks are tightly interwoven with how we use cars and other vehicles. An indication of the complexity and importance of the social side of the transportation infrastructure is the existence of driving schools and the effort it takes for parents to teach their children how to behave in traffic.

(Human) practice refers to established ways of acting towards a common goal based on shared understandings and explicit and implicit rules. They are supported by technical means (Dittrich, 2016; Schmidt and Simone, 1996). Thus, we regard software development as a

collaborative work practice supported by a material infrastructure of computers, network, and software. This perspective has been applied before: Draxler and Stevens (2011), for example, analyze Eclipse as a technical system tailored to and embedded in a team's collaborative development practice. They show how software teams continuously maintain their use and adaptation of Eclipse as their infrastructure as part of their daily development practices. (Draxler and Stevens, 2011; Draxler et al., 2014).

Software practices emerge from the daily work and experience of developers. Button and Sharrock (1994) describe how teams develop local practices based on their interpretation of company-wide formal software development methods. Fitzgerald et al. (2002) distinguish between *formalized methods* and *methods-in-action* – the developers' local and unique enactment of the formalized methods. With respect to both open source and proprietary software development Sigfridsson describes "The purposeful adaptation of practices" (2010, 2007). Dittrich (2016) argues that we need to understand software development as objectual practices that require a continuous adaptation of methods and tools to the software under design (Knorr-Cetina, 2001).

Infrastructures are not static. They change. Societal needs, vehicle designs, and the engineering of roads, evolve together with the written and unwritten rules of being part of traffic. Thus, infrastructures depend on *infrastructuring,* the continuous alignment, adjustment, and evolution of both the technical base and the social arrangements and protocols of an activity (Karasti, 2014; Karasti and Syrjänen, 2004; Star and Bowker, 2006). Infrastructuring has been widely used in the study of software development, especially to come to terms with the heterogeneous activities of software design and use (Dittrich et al., 2002) that can be observed in IT infrastructures supporting cooperative work in communities and organisations (Pipek et al., 2017; Pipek and Wulf, 2009).

The lenses of infrastructure and infrastructuring allow us to understand how CSE evolves through the interplay between changing organizational needs, evolving technologies, and improved practices and norms.

3. Research method

The study explores the motivation, context, and processes of CSE implementation and ongoing evolution. We designed a qualitative interview-based case study (Cresswell and Cresswell, 2018). In this section we first motivate the case selection and present the case companies, followed by a description of the interview design and the analysis process. Finally, we discuss the trustworthiness and limitations of the research.

3.1. Case selection and presentation

We contacted the companies and interview partners based on discussions at local industry networking events complemented by personal contacts. One of the authors had previously been involved in additional research in two of the case companies (see Table 1). The previous research was used to triangulate and corroborate the findings for two of the cases in the present study. The companies represent different kinds of industries and different kinds of software, have different sizes and structure their development in different ways. We present the companies and the interviewees below:

Table 1
Overview of interviews.

Company	Product	Role of Interviewee	Length of interview	Triangulating research
AccSaaS	Online accounting software	Developer CTO	0h56min 1h:24min	(Dittrich et al. 2020; Lous et al. 2018a; Lous et al. 2018b)
Rater	Product and service rating service	Developer Toolsmith Engineering director	1h:16min 1h:07 min 1h:21min	
ToolComp	Software installation tool	CEO & developer	1h:13 min	(Dittrich 2014)

AccSaaS develops and operates administrative software as a service. The company develops an online invoicing and accounting software solution for small businesses. It was founded in 2012 and has grown to 40 employees located in the headquarters and offices in two other European countries. It serves customers in 33 European countries. Fourteen out of the company's employees work in software development. AccSaaS's development team, at the time of the interview, was about to form sub-teams focusing on different parts of the software.

We interviewed the vice president responsible for the development, and a developer. One of the authors had earlier supervised a B.*Sc*. and an M.*Sc*. thesis at the company that resulted in several publications on distributed agile development and continuous evolution of software processes. The interviews complement the previous research focusing on CSE practices that have not been subject to earlier research.

Rater builds and operates a global platform for consumer reviews of companies and products. The company was founded in 2007 and has grown to 700+ employees in offices around the world. The development site within the company's headquarters had approximately 70 employees at the time of the interviews. The developers were divided into teams: A platform team, responsible for the product back-end platform, a DevOps team that builds and maintains development and deployment tools and stacks, and several product teams with responsibility for different parts of Rater's software products. The product and platform team structure reflects the software architecture.

Because Rater is larger and more diversified than the other case companies, we interviewed one of the tech leads, a developer, and a tooling expert. Also due to no earlier contact, we interviewed 3 members of the development part of the organization.

ToolComp began as a producer of GIS based applications for individual customers. The company has since then changed their business towards a system administration tool, (OpenSourceTool), that manages updates to operating systems and applications in local area networks. The first version of the tool was developed between 1995 and 1998. The customers are small businesses and organizations who cannot afford the competitors' more expensive off-the-shelf products. New versions of the tool are not pushed to production by ToolComp but pulled (downloaded) and installed by the users. ToolComp had 15 employees, out of which 11 worked in software development at the time of the interview. A core development team with 3–4 full-time and 2–4 part time developers is responsible for the ongoing development of OpenSourceTool.

ToolComp has implemented a development process inspired by Open Source that resembled CSE with respect to planning, implementation, unit tests and build.

The managing director, who was also one of the co-founders and core developers, was interviewed. He also presented the product at industry fairs and held user training workshops. One of the authors has followed the development at the company over several years. The company has been part of an earlier interview study on software product ecosystems. The new interview was executed to update the researcher about the recent development to include automatization of systematic testing.

Table 1 lists the characteristics of the companies and the interviewees and provides references to the publication of the triangulating research.

3.2. Data collection and analysis

We used semi-structured interviews for the data collection. The interview guide was designed around the activities in the continuous* framework (Fitzgerald and Stol, 2015) to learn about the business context, history, and state of CSE practices. The guide is included in Appendix A.

Each interview was conducted by two members of the research team, initially consisting of three researchers. The interviews took place at the premises of the companies. One of the researchers acted as the main interviewer, the other researcher took notes and supported with follow up question when some aspect did not become clear or a specific circumstance required further explanation. The interviews were recorded and a detailed content log for each interview was created by the researcher who had not taken part in the interview. An exception here is the interview with ToolComp that was held in German and summarized in English by the same researcher, being the only German speaker in the team.

Together, we performed a traditional thematic analysis (Robson and McCartan, 2016). Guided by the researcher who created the content log, the whole research team went through each of the interviews and assigned codes in an open coding manner. In this process we discussed how the interviews with the same company complemented, corroborated, or maybe contradicted each other.

This discussion benefited from prior research with two of the companies. The prior research allowed focusing the interviews on the recent changes respectively the specific topic and helped to interpret the interviews and transcripts, e. g., when the interviewee from ToolComp explained that they have developed software in a continuous manner for more than 20 years, this was understandable because the development process has been discussed in depth in previous interviews. Likewise, previous research on retrospectives in AccSaaS helped us deepen our understanding of their continuous improvement.

In the initial discussion, the codes were collected on a whiteboard and clustered. In this process the codes, their meaning and their application were discussed in detail and the understanding of the codes was harmonized. The consolidated coding scheme (initial and aggregated codes in Appendix B) was used to re-code the whole set of interviews. The re-coding was done by a different team member than the one who created the initial log. (Again, the German interview was an exception.)

The thematic analysis informed the development of the high-level themes we present in the initial analysis (Section 4). The full coding scheme is provided in Appendix B. Table 2 gives examples of how interviews were coded and aggregated to arrive at the high-level themes in the analysis (Section 4). The analysis includes rich descriptions and quotes to allow the reader to follow or contest our analysis.

During the analysis we realized that the thematic inductive analysis revealed but failed to explain the level of variation and the continuous evolution of practices reported by the interviewees. We, therefore, applied the theoretical concepts introduced in Section 2.4 to understand and describe the rationale behind the variation across the different cases. (See also Robson & McCartan's (2016) discussion of the role of theory.) The result of this theoretic, second level analysis is presented in Section 5 using both the original data and the findings in Section 4.

Table 2
Coding examples.

Interview	First level code	Aggregated code	Axial code (section)
[W]hen the Sprint finishes, you have to have everything in the done column before you can deploy the thing … and at one period we experienced a delay of 3 weeks in the done column. (AccSaaS, CTO)	Motivation for CSE	Motivation for CSE	Transition to CSE (4.1)
I think [the] architectural change has sped up the effectiveness of continuous delivery. … We always try to make things as easy to release as possible, and as small as possible.. (Rater, Engineering director)	Microservices	Architecture design	Product architecture (4.2)
One command and I can push stuff to production … [T]his command will run all the tests. If it passes it will go on to deployment. … It must be so easy to deploy to production, … but … to get there, there is a lot that needs to be set up. (AccSaaS, Developer)	Infrastructure for deployment	Infrastructure	Development and Deployment technologies (4.3)
You cannot just solve this [prioritization and balancing] through hiring, even if you had the money. [We] need the experience of the people who have done it for years. (ToolComp, CEO & developer)	Prioritization of tasks	Planning	Planning and management (4.4)
So [the coordination of changes] is always a bottleneck, but that is a necessary one, because the team that owns [the context] needs to know what happened … It is like a mental synchronization between teams. (Rater, Toolsmith)	Inter-team coordination	Coordination	Coordination and Collaboration (4.5)

3.3. Trustworthiness and limitations

In this section, we highlight the aspects of the research design and implementation that especially support the trustworthiness of the research, namely case and subject triangulation, researcher triangulation, member checking, and rich descriptions.

3.3.1. Case and subject triangulation

The findings are based on three software development companies that deploy software in different ways and target very different sectors. The variation of the resulting practices and the rationality of evolution of the practices regarding both internal development and external factors was the core reason to refer to additional theory.

Different members of the companies were interviewed depending on company size and the diversification of roles and responsibilities. In the analysis, we first compared the findings from the interviews with the same company and then compared findings from different companies. Where applicable, the triangulation with prior research helped to deepen our understanding of the individual company's practices. We are convinced that the description of the companies and the findings presented below actually match the case companies' software development.

3.3.2. Researcher triangulation

The interviews and the analysis were conducted by a research team of first three and later two researchers as described above. We made sure that each of us either took part in or logged an interview. The open coding was done by the whole team together, and the coding scheme was intensely discussed throughout the analysis. The final thematic coding was then executed individually, and the results compared. Through this intense collaborative process, we made sure that we developed a shared understanding of the interviews and prevented individual biases influencing the analysis of the interviewees.

In recent years, qualitative empirical research in software engineering has begun to apply measures describing inter-rater agreements statistically, like e.g. Cohen's kappa coefficient (Cohen, 1960). This makes sense, especially for data sets that are too big to code by one person, if the coding needs to be done by research assistants to address potential biases, or if only one of the research team engages in the development of the coding scheme and the coding. The coefficient is normally computed based on a subset of the coded material. In the first two cases, the coefficient is used to calibrate the coding team and the coding scheme. In the latter case, the measure is used to promote the intersubjective interpretation of the coding scheme.

In the research presented here, the dataset was small enough to not have to rely on additional research assistants to code the data in a distributed manner. The close collaboration when creating the coding scheme assured the intersubjective understanding of the individual codes. A statistical measure would not have further improved the quality of the coding scheme nor the coding process.

3.3.3. Member checking

We were able to check our analysis results with interviewees from two of the companies. The interview partners confirmed our understanding of the two companies' analysis.

3.3.4. Rich descriptions

The findings section refers to the details of the software development practices related by the interviewees and includes a substantial amount of verbatim or translated citations from the interviews. The level of detail allows the reader to follow the researchers' analysis all the way from the interview to the categories used in the analysis.

3.3.5. Limitations

Our analysis is based on 6 interviews in three companies. Qualitative research does not aim to create externally generalizable results, however. We nevertheless trust that the triangulation across different companies and industries, together with our careful analysis, results in insights that are relevant beyond our case companies. To strengthen the transferability of the findings and insights, we discuss them in relation to related empirical research in Section 6. Further, relating the findings to theoretical concepts, in our case infrastructuring, provides a basis to discuss transferability of the insights to other contexts. This will be further discussed in Section 6.

The internal validity of our findings is supported by the triangulation across different interviewees with different roles in the companies and by earlier and parallel research with two of the companies.

4. Analysis

In this section we present the findings from the analysis of the case organizations. The analysis is structured according to the themes we identified in the thematic coding: the transition to CSE, (changes to) the product architecture, development and deployment technologies, planning and management, and coordination and collaboration. We strive to provide a rich description of each case through separate sections for each case company within the themes. In this description, we relate the CSE related change to other analysis topics as well as to the company and the business context. The motivation is to show the rationality specific to each case, which in turn motivated us to pull in theoretical concepts from outside software engineering to understand the variety and evolution of the practices. There is a cross-case summary and comparison in each theme section.

The last subsection, Section 4.6, provides a summary of the analysis and a cross-case comparison, highlighting core commonalities across the three cases – a continuous development process, empowered development teams, the emphasis on a development, test and deployment infrastructure supporting the flow of finished features into deployment – as well as significant differences between the cases regarding the implementation and evolution of practices and tools, and the way to organize the development and deployment. We relate the differences to the type of software product, the business model, and the company's specific history. Table 3 summarizes the findings. The coding scheme used for the analysis of the interviews can be found in Appendix B.

4.1. Transition to CSE

4.1.1. AccSaaS

AccSaaS's transition to CSE began a little more than three years before the interviews. It was motivated by delays and lack of flexibility when deploying large product increments after a Sprint:

" [W]hen the Sprint finishes, you have to have everything in the done column before you can deploy the thing … and at one period we experienced a delay of 3 weeks in the done column." (AccSaaS, CTO)

The idea to release in smaller increments came from a developer and was immediately taken up by management who drove the change. The company already used Github and Jira for configuration and task management, and had implemented automated testing, and one-click deployment. This allowed them to establish the technical infrastructure for continuous integration and deployment in 2 weeks, but it took more time to train developers and redesign the development and deployment processes for CSE:

"What we did not have was like all the people process, and working in parallel, and we had not started working in parallel in the beginning …" (AccSaaS, CTO)

4.1.2. Rater

Management initiated the transition to CSE four years before the interviews. The motivation was to shorten the agile release cycles of 3–4 weeks and simplify configuration management and code integration through smaller releases. The company also aimed to break up the monolithic legacy architecture that had evolved over the years into small, decoupled components, i.e., microservices.

"We had like 3-week sprints, SCRUM, and a release more or less like every 3 or 4 weeks. (…) Also the code was one big system, interconnected, in one big solution. (…) And there is a lot of risks with each release, because everything was interconnected." (Rater, Engineering director)

The transition to a microservice based product architecture is not complete but the CSE processes and practices are in place, and Rater deploys new versions 150–160 times per week (Rater, Engineering director).

4.1.3. ToolComp

With respect to the implementation of continuous planning, implementation, unit test and built, ToolComp implemented practices resembling CSE from the very beginning. ToolComp's current change to their process was motivated by a need to streamline the integration test process for OpenSourceTool. The original integration, test and delivery process combined automated unit tests and integration with manual test and release activities in a staged release process. The final testing stage also required the active involvement of the system administrators at the customer sites. Increased diversification and release frequencies of the operating systems that the tool manages motivated ToolComp to take steps towards a faster – automated – integration and test process:

"The change [towards automated integration and testing] is not triggered by changes in the software development, but more because our context became more complex: We simply have gotten new Window versions in rapid succession in the last years. [We] now … support Linux based target machines, they also tend to release new versions frequently." (ToolComp, CEO & developer)

The shift to automated testing would also reduce the need for customer involvement in the final test phases (ToolComp, CEO & developer)

4.2. Product architecture

The monolithic architecture typical of traditional client-server software architectures is unfit for frequent deployment due to the build and test effort required for even small changes. The transition to CSE is, therefore, often accompanied by a refactoring of the software product into separate components, such as micro-services, with low coupling and well-defined interfaces.

4.2.1. AccSaaS

The company has worked towards replacing the monolithic architecture with microservices since the beginning of the CSE transition, (AccSaaS, CTO). The refactoring is not – and may never be – complete, however, because the perceived benefits, i.e., shorter integration and delivery cycles don't justify the cost and effort required to complete refactoring the software.

"[Q]uite a big chunk of the domain model is one service. I would like to split that up … but it is well structured code. [It] is not high on the priority list to break that up. … It takes a bit long to run the tests. the main difficulty is not maintaining it, but that running the tests is slow." (AccSaaS, CTO)

4.2.2. Rater

Rater initiated the change towards microservices with APIs and separate databases to enable continuous delivery:

"I think [the] architectural change has sped up the effectiveness of continuous delivery. … We always try to make things as easy to release as possible, and as small as possible…" (Rater, Engineering director)

The change has taken a long time and is not complete because of the effort needed to refactor the back-end components:

"[While the transition] was easy for the new features or the smaller products … this was harder for what has been developed in a single repository … . So, we have taken out [parts of the software] … to a proper … microservices architecture. And this was the first project … and it took one year to complete." (Rater, Developer)

It will take a year at least to complete the refactoring of the old software and in the meantime the developers must work with both architectures (Rater, Toolsmith).

The microservices architecture introduces challenges of its own. It supports independent and parallel development and deployment, but features that cross boundaries between services create new dependencies between components and teams:

"[[D]ifferent teams own different contexts, and … no other team would access another context's database directly, they'd go to some API. (…) They are now the customer of that team." (Rater, Toolsmith)

The microservices architecture impedes access to data across different services for analytical purposes, and the company has had to create shared storage – a data lake – to ease access to product-wide data:

"[G]etting data from different data bases into a single place … almost takes a developer, because you need to call APIs with different permissions and different types of authentications … And the data lake is sort of moving towards it … we start shoveling this into the data lake and we are just getting started on it." (Rater, Toolsmith)

Finally, individual team ownership and low coupled code depends on team coordination to secure overall product coherence and avoid redundant code:

"Being independent and … this code ownership model means that [the services] sometimes drift in opposite directions … [I]t can also be dangerous of course if everyone has their own library to do the same thing. So, knowledge sharing and alignment across teams, that is actually a big challenge." (Rater, Engineering director)

4.2.3. ToolComp

New versions of OpenSourceTool are not deployed automatically but downloaded by system administrators at the customer sites. Rapid deployment is, therefore, not needed, and the test automation did not trigger major architectural changes. However, the product has evolved towards a large and complex cross-platform server-side application that manages software installation on several clients with different operating systems and versions. New versions of the product must be tested against combinations of client and server-side platforms and operating systems, which complicates the automated integration and test set-up. Further, the tool relies on a Linux kernel and must be retested regularly to assure that new Linux versions do not break the software.

4.2.4. Product architecture summary

The CSE transition is different in breadth and depth in the three companies. Changes to ToolComp's OpenSourceTool are driven by changes in the product's run-time environments, i.e., the operating systems the tool manages, rather than requests for new or enhanced functionality. The OpenSourceTool architecture has, therefore, been more stable than in the other two cases but changes are now needed.

AccSaaS and Rater initiated a comprehensive refactoring based on microservices, but the product architecture may never reach a uniform and final state because the effort needed to complete the refactoring exceeds the benefits, and the developers must work in both architectures for the foreseeable future.

The service-oriented architecture reduces dependencies in the code and shortens the deployment cycle, but it also introduces new challenges and workarounds. New features can cross established service boundaries and create new code and data dependencies, and the lack of a shared database introduces redundancy and impedes extraction of data for analytical purposes.

4.3. Development and deployment technologies

Frequent and rapid deployment of software depends on fully or partly automated build and test technologies, as well as swift allocation of run-time environments for new software. Such pipelines are not unique to CSE and they were used in the agile – timeboxed – development and deployment prior to the companies' transition to CSE. The challenge in CSE is to integrate existing components and processes into a complete and stable continuous deployment pipeline.

4.3.1. AccSaaS

The basic infrastructure for automated deployment was established prior to the shift to CSE as described above. The automated deployment process specifies the components included in the build, (regression) test cases, and the production environment. Previously, this included all or most of the separate front- or backend code and test cases but the microservices architecture uses separate deployment pipelines for different parts of the product. This shortens the deployment due to smaller builds and fewer test cases, but it also requires addition of new pipelines when new services are introduced. This is a complex and time-consuming process:

"One command and I can push stuff to production … [T]his command will run all the tests. If it passes it will go on to deployment. … It must be so easy to deploy to production, … but … to get there, there is a lot that needs to be set up." (AccSaaS, Developer)

Control over the deployment and operations environment has shifted over time. The deployment infrastructure was originally outsourced, but the service provided by the external partner was unsatisfactory, and AccSaaS decided to backsource operations.

"The problem we had been waiting for months … we fixed in a day. That was a lesson learned … that whole system of continuous integration server … and all of that … should be run by someone very close to the team." (AccSaaS, CTO)

4.3.2. Rater

Rater used a .Net based deployment infrastructure prior to the transition to CSE. Issues with this technology has prompted a transition to Docker. The Docker environment enables shorter set-up and boot

times and removes the separation between the development and production environments by allowing developers to work in a local version of the production environment:

"Now, with Docker, you can say: it is working on my machine with production configuration and that actually means something. That is enormously valuable, … because you can essentially completely mimic a production system…" (Rater, Toolsmith)

The transition to new tools and environments takes time, however, and comes with its own set of issues. First, the hybrid product architecture – with the remains of the old monolith and the new microservices architecture – causes Rater to maintain several development and deployment environments:

"[The old core] has become massively smaller, but … we are still talking … maybe a year, before it's all completely gone. But until then, we just have three different deployment systems and all this stuff." (Rater, Toolsmith)

The DevOps team has had to step in and carefully help developers who are used to the .NET environment become confident with the Docker platform and learn how to use it in the best way:

"[Introducing Docker] comes … down to making people comfortable, like taking the discussion, educating them … [Y]ou can't go too fast, because the second you go too fast, … you start leaving people behind." (Rater, Toolsmith)

4.3.3. ToolComp

The CSE effort is directed at automating integration and testing as much as possible with test scripts. (ToolComp, CEO & developer). The integration and test for OpenSourceTool requires several (virtual) client-server networks and testscripts to test the tool on different operating system combinations. At the time of the interview, scripting was finished for the retest of stable versions of the software but was still under development for the development pipelines. New operating systems versions and changes to OpenSourceTool itself results in changes to the deployment pipelines and there is an increased focus on ensuring the testability of the changes to the tool:

"When you design the software, design for testability …. […] Is there a command line function? Is there some output that I can enforce? Are there other places, where I can steer the thing in order to create a unit test, as well as an integration test. Testability [has become] a very explicitly formulated requirement." (ToolComp, CEO & developer).

4.3.4. Development and deployment summary

The three case companies have implemented automated 'one button deployment' to varying degrees. But the development and deployment infrastructures are neither perfect nor stable. New features or other changes to the product require manual set-up of new integration and test scripts. Changes to development technologies and paradigms furthermore require new development and deployment platforms and pipelines parallel to the older platforms and pipelines still in use.

We find it unlikely that these companies will ever arrive at a stable, unified development and deployment architecture and pipeline: Multiple development and deployment technologies are needed to support the hybrid product architecture, the removal of which is not highly prioritized in AccSaaS and Rater At the same time, these companies introduce new technologies to overcome the limitations of the service-oriented architecture and developers must build skills and adapt practices to work with the new technologies.

At ToolComp, on the other hand, the increasingly heterogeneous operational environment and rapidly evolving target operating systems necessitate the development of test automation environments that can manage both regular retests for new versions of operating systems and dependencies, and more systematic development tests.

4.4. Planning and management

CSE – like agile software development – uses an adaptive approach to planning with flexible releases and deadlines but replaces timeboxes and Sprints with a continuous flow of tasks. In this section we explore the planning and management processes in the case companies.

4.4.1. AccSaaS

Development planning at AccSaaS is adaptive and informal. The CEO, the CTO and the PO meet to discuss product features and other issues every second week. The PO and CTO prioritize tasks and balance developer resources at informal meetings. Long-term release planning is kept to a minimum and detailed planning decisions are made on an as needed basis. Task assignment aims to balance the load on the developers. All developers can work on all parts of the product. (AccSaaS, CTO).

The developers groom larger assignments and divide them into manageable subtasks, which they release when ready. This approach generally works well but there have been cases of a developer getting stuck on a task that was more complex than first anticipated. Therefore, tasks that are estimated to take more than two days are assigned to two developers who monitor each other's progress:

"As soon as you do anything that is more than a couple of days work, we always put two people on it and then they will notice [if the other person] gets stuck." (AccSaaS, CTO)

AccSaaS has eliminated the QA function and shifted the responsibility for product quality to the developers to increase the rate of delivery instead of spending resources on defect prevention and correction:

"[The feed-back from our customers showed that] we have no quality issues, we need to build features." (AccSaaS, CTO)

The CTO feels confident that the developers will live up to their responsibility for the quality and stability of the software.

" [Y]ou write that extra unit test, or you write that extra integration test because you want to make sure … you [don't want to] push this to production and then it goes 'boom'." (AccSaaS, CTO)

The developers appreciate the trust, but it took them some time to build the confidence and knowledge needed to become "comfortable touching production" (AccSaaS, Developer).

4.4.2. Rater

Planning is high level and flexible (Rater, Engineering director). Strategic goals set by management drive the development of new features on the platform, but they are not translated into firm release plans.

"Well, we don't say anything about any products we release, right. We don't promise anything. (…) The business goals [are] our focus of course. We promise to look at that and make that a priority." (Rater, Engineering director)

The high-level goals – e.g., attract more new customers – are broken down into quarterly Objectives and Key Results (OKR) and further into epics and stories in collaboration with the team in charge of the context (Rater, Engineering director). The use of estimates is limited to the final break-down of features into tasks:

"[W]e try to break things up into small parts, like a day or less, and branches should be less than a week old." (Rater, Engineering director)

The team and the tech lead agree on the implementation details:

"You agree in the team on the direction. … So, we should put in an architecture that is this and this and that. And we should start by moving this piece first." (Rater, Developer)

Management deliberately protects the team from strict deadlines and targets to avoid excessive pressure:

"Sometimes, we set, like, high-level deadlines, like, saying, ok, we want to have this done by June, or something like that. But … that is not really something that hits the teams [it] is managed by management." (Rater, Engineering director)

The development teams are responsible for product quality and testing:

"[T]hey are expected to do both. … That means there is a little bit difference on … how much the teams do test, but they are ultimately

responsible for their (…) uptime. So, it is in their best interest to spend some time on testing." (Rater, Engineering director).

4.4.3. ToolComp

ToolComp does not create a release plan as such because most software updates are triggered by operating systems releases and requests from customers needing support for changing hardware configurations. Interestingly, the number of the latter has increased due to the ongoing systematization and automation of testing, which reduces the work needed to test the software against new hardware.

"[The customers] do not have to do strange workarounds, when they buy new hardware, because we keep the things up to date with this rolling release, and therefore we can quickly react to new hardware." (ToolComp, CEO & developer)

Bug fixes and refactoring comprise the last group of development tasks.

The developers discuss and prioritize tasks at weekly developer meetings with all technical people in the company, including those working with support. The meetings focus on product development and backlog issues as well as how to address more complex tasks, but they are also used to discuss the development of the new integration and test environment.

Prioritization and distribution of the different tasks among ToolComp's experienced developers is a concern:

"You cannot just solve this [prioritization and balancing] through hiring, even if you had the money. [We] need the experience of the people who have done it for years." (ToolComp, CEO & developer)

4.4.4. Planning and management summary

The case companies manage product development at different organizational levels and with different time horizons. AccSaaS and Rater base their plans for the long-term evolution of their products on company strategy, market analyses and customer feed-back. Overall product development, targets, tentative release plans, etc. are managed at the top management level.

ToolComp does not have strategic and long-term plans for OpenSourceTool. The evolution of the product is primarily driven by changes to the products' context outside ToolComp's control, i.e., changing operating systems and hardware configurations.

Short term planning is in the hands of developers and their managers: Product changes as well as technical tasks, e.g., adaptations to new development and deployment technologies, refactoring, and bug fixing, are broken down into development tasks (groomed), prioritized, and distributed among individual developers or teams in collaboration with the developers. All three companies emphasize the need to balance new development and product maintenance as well as the developers' workload in this process.

4.5. Coordination and collaboration

Coordination and collaboration are needed to establish and maintain shared development and deployment practices among teams and developers and resolve issues that cross the boundaries between different parts of the code.

4.5.1. AccSaaS

AccSaaS's developers share responsibility for all parts of the code. The company Wiki describes the architectural framework, coding standards, and patterns, but the developers must still coordinate to resolve dependencies and other issues in the code:

"Because we are … free to do stuff as we see fit, also as developers, we constantly have to be in dialogue … agree about how we do stuff. … There is … some pain in always … getting to a consensus… and that is always a little bit hard. But it is important." (AccSaaS, Developer)

The move to microservices supported independent testing and deployment of features, but these benefits did not materialize, however, because of frequent test failures and crashes. The developers had to agree on a new test and deployment process to solve the problem:

"[Y]ou need to run a lot of services to run the test and sometimes they crash and then tests can't run … [P]eople were waiting to be able to push to master and production. That changed when we changed the way we do … feature requests … test had to run green before being integrated." (AccSaaS, CTO)."

AccSaaS's software development changes over time. The developers reflect upon and suggest improvements to their development and coordination practices at biweekly retrospective meetings. Emphasis is put on experimentation with and evaluation of new ways of working.

4.5.2. Rater

The developer teams at Rater are responsible for their own 'context', i.e., the code and data they manage. This enables rapid and parallel implementation of changes but depends on coordination between teams to ensure the overall integrity and quality of the code. This takes place at several levels: First, at the organizational level, the tech leads and the engineering director meet regularly to oversee the general product development trajectory and ensure that the work of one team does not "break … what other teams are doing technically." (TP, Developer).

Second, at the inter-team level, development teams coordinate changes that cut across team contexts. The team that owns a context are responsible for all changes to their code. They will implement changes requested by other teams, or inspect changes made by others before they are committed to the delivery pipeline. This can delay the implementation of needed changes but secures the integrity of the code and supports important knowledge sharing between teams:

"So [the coordination of changes] is always a bottleneck, but that is a necessary one, because the team that owns [the context] needs to know what happened … It is like a mental synchronization between teams." (Rater, Toolsmith).

Third, Rater has established a DevOps team responsible for the development and deployment platforms and pipelines. The team oversees and maintains the development and deployment infrastructure, and ensures that developers follow the standards and processes mandated by the company or needed by the technical infrastructure:

"They [the DevOps team] are the touch point, when we need something, … The architecture we want to go for this and this and this, therefore we need this stack, this stack, this stack, these servers, these queues, or what not." (Rater, Developer)

This requires frequent consultation between developers and the DevOps team (Rater, Developer), as well as instruction and training by the DevOps team (Rater, Toolsmith).

Finally, development teams meet regularly to align the work of individual developers and agree about details of the development tasks:

"Do we agree on what these things are? How they should work? And how they should be built within the team." (Rater, Engineering director)

The teams at Rater also strive to improve their practices and perform biweekly retrospectives where they suggest improvements to issues raised by developers. The results are evaluated at future retrospectives.

4.5.3. ToolComp

The 3–4 core developers at ToolComp coordinate product development during meetings and day-to-day interactions. Weekly meetings between all developers and technical staff function as the main knowledge sharing hub and the primary communication and coordination mechanism in the company. The meetings are used to prioritize entries in the issue tracking system, and track the status of ongoing development tasks and the improvements to the test automation environment.

Most changes to practices are triggered by the deployment of new tools and grow out of the day-to-day work in ToolComp's small and tightly knit developer group. Changes or issues that require more focused discussion are taken up at the weekly developer meetings.

4.5.4. Coordination and collaboration summary

CSE emphasizes developer and team independence where developers and teams can implement, release, and deploy increments to the product independently from another. Our three cases show that this independence may not be fully achievable in practice but requires ongoing coordination at different levels.

At the product level developers and teams working on the same part of the product agree upon the details of the code being created, and separate teams negotiate/coordinate when new features cut across team or component boundaries.

At the architectural and infrastructure level the companies define and enforce standards and train developers in how to use them. These standards evolve with the introduction of new development and deployment tools and platforms.

Finally, at the organizational level, managers oversee and coordinate the direction and pace of overall product development.

The three case companies organized the coordination and process improvement work differently. All companies let the developers resolve issues with the product and processes – sometimes in collaboration with management. ToolComp and AccSaaS also let developers build and maintain the deployment pipeline and processes, while Rater used a separate DevOps team, which was also responsible for training the developers.

4.6. Case analysis summary

The overview of findings (Table 3) reveals similarities as well as differences among the three cases: First and foremost, all three companies aimed to support frequent deployment of new versions of their software product in a safe way. To this end all companies invested and continue to invest in what is often called 'the pipeline', automation of integration, test and deployment or distribution. This, in turn, requires changes to the software architecture. All case companies empowered development teams to decide how to develop software. At the same time, however, the development practices reported by the interviewees differ to the extent that we might not be able to talk about a common transition path towards CSE. Tools and platforms, standards, guidelines, and explicit rules may be in place, but they are complemented by direct interaction and (informal) agreements among developers and teams.

We furthermore observe that the companies did not reach a complete and stable implementation of CSE despite several years of change; in the case of ToolComp for more than 20 years. Rater and AccSaaS had not completed the refactoring of the software product architecture, and ToolComp struggled to automate the test. At the same time, new technologies and platforms, requests for new features, and inadequate practices compelled developers in all three organizations to adapt their practices continuously. Rater relied on team and inter-team coordination together with a new unit, the DevOps team, responsible for this work, ToolComp assigned a developer, and AccSaaS' developer team continuously adapts work practices and the technical infrastructure as part of their development work.

We analyze the primary findings through an infrastructure lens in the next section.

5. CSE as infrastructure

None of the case companies had completed the transition to CSE but continuously adapted the technology, the product architecture, and the development practices to changing circumstances. Further, the implementation of CSE at each company was supported by local rationales and circumstances. These observations motivated us to apply the theoretical framework from Section 2.4 to a second level analysis of the cases to make sense of the continuous evolution of CSE we found in the companies.

5.1. Infrastructures in CSE

The physical or technical software infrastructure – the software architecture and the development and deployment platforms and tools – enables continuous deployment of changes to the software product in a controlled and safe manner. It provides different ways to develop and deploy software.

The effective use of the technical infrastructure depends on the social infrastructure, i.e., the organizational structure, and the development practices and norms – the social protocols – agreed upon and enacted by the developers and their managers. It follows that a CSE infrastructure is a specific and local configuration of technologies, organizational structure, and practices (see Section 2.4). We can, therefore, relate the different paths to CSE that the three companies chose, to their specific infrastructures.

Table 3
Summary of analysis findings.

Company and Software	Transition to CSE	Product Architecture change	Development and Deployment technologies	Planning and Management	Cooperation and communication
AccSaas Economy software as a service. 14 Software developers working distributed, mainly in Denmark and the Ukraine.	Moved from agile to CSE by introducing continuous integration and build, which resulted in more independence in the development.	Changing a monolithic architecture to microservices. Monolith legacy software partly refactores to reduce integration and test time.	Development and deployment infrastructure was established before CSE. Responsibility and the way the infrastructure was used changed.	Adaptive bottom-up planning; the team decides on the tasks, grooms them and breaks them down.	Emphasis on consensus building; Tooling adjusted to support independent and parallel work Evolving practices and roles.
Rater Service software for rating enterprises. 700+ employees, ca 70 developers at the Danish site.	Motivation for CSE: Shorten the release cycle.	Move from monolithic architecture to microservice architecture. Not completed New focus on data integration of data for more comprehensive data analytics.	New (Docker) and old (. Net) environments with separate deployment pipelines co-exist.	Strategic goals set by management, but without set release dates. Teams responsible for task breakdown and scheduling.	Cross disciplinary teams; Coordination between teams and between teams and product manager through the tech lead. Code ownership model supports coordination between teams DevOps team maintains technical infrastructure and guides practices.
ToolComp Tool for installation of software in a LAN. 15 employees of which 11 are working with development.	Development process already CSE due to open-source tradition. At the time of the interview: introduction of test automation due to changes in the environment.	No major change as the test automation does not change require architectural change in the moment.	No real change to the development. The changes to the test automation implies a major development effort. New emphasis on testability.	Requests for new features to accommodate changes in the hardware and operating systems originate from customers. The main planning activity is the developer meeting.	The core developers coordinate work and set objectives during developer meetings and in day-to-day interaction.

AccSaaS's and Rater's transition focused on the development process and product architecture. Both organizations had most of the technical components in the integration, test, and deployment pipeline in place before the transition, but they had to refactor their software product architecture to reduce the integration and test effort needed to deploy changes. The Sprint based development *process*, however, caused bottlenecks and delays in deployment. These bottlenecks disappeared when the companies moved to continuous development and deployment. Together, these changes allowed the companies to realize the immediate benefits of CSE.

The two companies have organized their development teams differently, however, and there are variations in the norms and agreements regulating the developers' work: AccSaasS single development team coordinated internally, while Rater's independent teams relied on formal inter-team processes and agreements to coordinate work and maintain the integrity of the independent services.

ToolComp evolved their technical infrastructure with a focus on test and integration technologies and processes. OpenSourceTool's original development *process* already resembled continuous rather than agile development. Updates and changes were deployed on an as needed basis, but the integration and test processes and technologies could not support the short release cycles required to response to the increasingly rapid changes in the tool's run-time environment. Thus, the transition originally prioritized improvements to the integration and test infrastructure, while changes to the software product came later.

5.2. A continuous transition

The three companies are in many ways still 'in transition' towards CSE. AccSaaS and Rater have not completed refactoring the legacy software, but this is not highly prioritized. The companies prefer to maintain a hybrid product architecture and 2–3 parallel deployment pipelines to spending resources on a complete product redesign with limited effects on the release process. Automated tests are still work in progress at ToolComp.

More importantly, internal changes as well as changes in the companies' environment drive continuous evolution of the CSE infrastructures in the companies.

Development and deployment technologies change. At Rater, the newer Docker environment was introduced to replace the old .NET and Visual Studio platforms. Docker provides better DevOps integration, but it changes programming and the deployment pipeline, and not all developers are comfortable using it.

Needs and requirements change over time: AccSaaS and Rater refactored their software with decoupled microservices and separate databases. New features do not respect the boundaries in the new architecture, however, and new dependencies arise. Therefore, new technologies and concepts, such as Data Lakes, are introduced to overcome the strict separation of data between services. ToolComp initially focused the CSE transition on integration and test: changes to quality assurance were needed due to frequent changes to the targeted operating systems and the Linux kernel. These changes in the environment caused more frequent updates to the software.

The companies strive to ***continuously improve processes***. Developers and managers in all companies regularly reflected upon and improved the development and deployment processes. Either through informal meetings within or between teams, or at formal meetings such as, e.g., retrospectives.

The changes originate in the technical as well as the social/organizational domains, but they affect all aspects of the CSE infrastructure in the companies. The introduction of a new technology – Docker – at Rater meant that the developers had to learn new development practices. Rater's developers also had to agree on new collaboration and coordination protocols in response to evolving requirements. And increased frequencies of changes to the operational context of OpenSourceTool caused the company to also address the product architecture as part of their CSE strategy.

We don't expect the companies to ever complete their journey towards CSE. New technologies, internal and external changes, and continuous improvement of practices mean that they will always be 'in transit' towards CSE.

5.3. Infrastructuring

CSE infrastructures don't change by themselves. Their evolution and maintenance depend on human work – 'infrastructuring' (see Section 2.4) – to manage and maintain the development and deployment technologies and tools, as well as negotiate *and* renegotiate the rules and standards for development and deployment. The three companies have developed different ways of how, when and where infrastructuring takes place: At AccSaaS, biweekly retrospectives among developers together with continuous experimentation with new processes, tools and techniques were the core instrument to keep the infrastructure aligned with the team's and the company's needs. At Rater, each developer team maintained and evolved team level norms and practices, while formal processes and inter-team collaboration secured the alignment between teams. The 'DevOps' group, further, was responsible for the maintenance of the technical infrastructure and its alignment with development practices. At ToolComp, infrastructuring is part of the weekly development meetings. Development of the technical side of the infrastructure is an explicit task along other development tasks taken on by the core development team.

5.4. Summary

The infrastructure lens enables us to understand how the heterogeneity of the observed transition history, current practices, and ongoing changes can be seen as instances of similar patterns:

All cases aim at reducing time to deployment, and provide feedback between use, operations, and development, but the characteristics of the resulting development practices vary due to different contexts and the close feedback loops.

The interviewees all reported on continuous adaptation and evolution of their process and infrastructure. We also found explicit infrastructuring in all three companies that helped to deliberate and implement the changes.

The next section discusses the implications of our findings to the understanding of CSE and software development methods and processes in general.

6. Discussion

In this section, we discuss our findings regarding the implementation and on-going adaptation of the CSE processes and technologies in the case companies. In the final section we discuss the implications of our study for research and practice.

6.1. Introducing CSE

Our case companies introduced CSE at the initiative of software developers and/or managers aiming to increase productivity and reduce delays in the CI/CD pipeline as reported by, e.g., (Klotins et al., 2022). We also found that the ability to continuously integrate, test and deploy software required comprehensive, and evolving changes to the software product, organizational structures, developer skills, and development and deployment technologies and practices as reported by, e.g., (Elazhary et al., 2021; Hemon et al., 2020; Klotins et al., 2022; Leite et al., 2021). Finally – like Klotins and Peretz-Anderson (2022), we show that companies select the path to CSE that fits their needs and context.

We make three contributions to this research stream. First, most studies of CSE comprise only some of the CSE activities in Fig. 1 (Klotins and Gorschek, 2022). We extend these studies by studying the complete

pipeline from development to deployment. Second, we show *how* companies choose their own paths towards CSE. There were common elements in the overall transition, such as, e.g., refactoring legacy software, but each company followed its own transition process depending on organizational context and rationale for the transition to CSE (see Table 3 and Section 5.1 above). Third, while there is a rich literature about CSE introduction and practices there are only few empirical studies of the socio-technical aspects of CSE (Klotins et al., 2022). Using our infrastructuring lens, we show in Section 5.1 how organizational goals interact with social and technical infrastructures to shape the transition process: AccSaaS and Rater wanted to increase their deployment frequency and focused the initial transition on the development process and product architecture. ToolComp, on the other hand, aimed to reduce the effort needed to integrate the OpenSourceTool into their customers' operational environment and started the CSE transition with improvements to their testing and integration technologies and processes.

6.2. Evolving CSE. A never-ending story

The companies have not completed their journey towards a complete CSE process, and they might never reach a stable CSE state. First, similar to findings reported by, e.g., (Klotins and Gorschek 2022) the companies focus on the software development-to-deployment pipeline and do not pay much attention to other parts of the continuous* cycle, i.e., the continuous feed-back loops between software development and the business (Fitzgerald and Stol 2015). Second, as also observed by e.g., Laukkanen et al. (2017) and Leite et al. (2021), the companies have not – and may not ever – completed refactoring the software product. The organizations must, therefore, maintain old development platforms, deployment pipelines, and practices alongside the new (cf., Leppänen et al. 2015).

Importantly, our analysis in Section 5.2 shows that CSE technologies, organizational structures, and practices are unlikely to reach a stable or final state but undergo constant change. O'Connor (2017) examines how changing technologies drive continuous change of CSE. Through the infrastructure lens we show that the CSE process adapts to emerging technologies as well as to changing product requirements, and new organizational needs. Thus, we also contribute to understanding the experimentation and innovation process in continuous* (Fitzgerald and Stol 2015).

6.3. Infrastructuring

Maintenance and improvements to the development and deployment infrastructure is an important and prominent part of CSE, as shown above in Section 5.3. The automated build-test-deployment process expected in CSE is neither perfect nor complete but depends on human work to 'close the gaps' in the technical platform, when, for example, a complicated manual process is required to set up a new deployment process and production environment (AccSass).

Thus, CSE contributes to a more efficient and automated development and deployment process, but additional work – infrastructuring – emerges to adapt work practices to shifting requirements and needs and align them with the technological infrastructure. In other words, while the technologies underlying CSE automate the work of operations, i.e., preparing and managing the operational environment, new work and organizational structures have evolved to coordinate and maintain the development and deployment infrastructure.

The importance of constant monitoring and adaptations to the technical and organizational CSE infrastructure is recognized in CSE research, cf., the platform team (Leite et al. 2021) or the build sheriff (Shahin et al. 2017). The infrastructuring in our case organizations was, however, an ongoing activity at all organizational levels and functions, including but not limited to, a dedicated organizational unit or role as we describe in Section 5.3.

6.4. Implications for research and practice

For both research and practice, the work presented here confirms a change in the understanding of how software practices evolve (Dittrich, 2016; Fitzgerald et al., 2002). When transitioning to CSE, methods, elements thereof, techniques and tools serve as inspiration, but they do not determine how a software development organization adapts and combines these components to the specific organizational context, its software products, technologies and development practices.

For research on CSE, the concepts of infrastructure and infrastructuring open up new opportunities to study the evolution and maintenance of tightly interwoven software engineering processes, practices, tools and techniques. Instead of discussing the application and limitations of technologies and methods, infrastructure and infrastructuring provide a lens to study the adoption and adaptation of method elements as a skillful activity. The infrastructure concept relates the technical and social dimensions of software engineering, e.g., how specific affordances in the tools we develop require certain processes and social protocols. The focus on how methods are adopted and adapted in specific situations and, especially, the rationale thereof will further provide important knowledge allowing to specify the usefulness of methods and methods elements in relation to specific contexts and improve the usefulness and usability of methods, processes, techniques, and tools.

The infrastructure evolves as contexts and requirements change and continuous infrastructuring becomes an integral part of CSE. Because of their importance, infrastructuring practices become more prominent both as the activity that links espoused methods and methods in use (Fitzgerald et al., 2002), and as the core of the adaptation of software development practices to internal and external triggers of change as reported in our analysis.

What furthermore becomes visible is that the core skill of software engineers is not the knowledge of specific methods or specific tools and techniques, but the ability to adapt and adopt tools and techniques in a flexible way; develop explicit rules and implicit social protocols to underpin their usage; and to be able to continuously develop the infrastructure that underpins continuous development and deployment.

Our findings are based on only three companies and they focus on the development-to-deployment pipeline in the continuous* cycle. These were the primary activities in the case companies' transition to CSE. The companies, furthermore, did not face regulatory or other constraints limiting their adoption and adaptation of CSE. However, like other research, e.g., (Elazhary et al., 2021; Klotins et al. 2022) our research shows the importance of the local context for the adoption and continuous adaptation of CSE. Our application of infrastructure and infrastructuring allows us to uncover how the adoption and adaptation takes place. Further research in other settings and other parts of continuous* is needed to supplement and extend our results.

In this research we have used infrastructuring to understand *how* the organizations implemented and continuously adopted CSE; including how they solved the frictions and setbacks encountered through both technical and organizational means. Further research is needed to understand the motivations, organizational challenges, costs and benefits of the implicit and explicit decisions made in the process (Claps et al., 2015; Klotins et al., 2022; Klotins and Talbert-Goldstein, 2023).

For industry, the research results emphasize the importance of a continuous evaluation and evolution of own practices, especially in the context of CSE. Be prepared for a 'never ending story' as our title suggests: The continuous evolution and improvement of development practices, the fostering of responsibility and trust between management and development teams, and the establishment of 'practices of changing practices' (Dittrich et al., 2020), e.g., in the form of retrospectives, are important to regularly reflect, innovate, and experiment with the team's way of developing software.

7. Conclusion

We began this research asking how software organizations initiate and sustain CSE practices, processes, and technologies. To understand the continuous work with processes, organization, and technical infrastructure, we introduced the concepts of infrastructure and infrastructuring as they are discussed in Information Systems and Computer Supported Cooperative Work. CSE depends on sociotechnical infrastructures consisting of a tightly coupled configuration of processes, practices, organizational roles, and tools, and software architecture patterns. The infrastructure needs to be continuously adapted as a reaction to changes in the usage of the software, the tools and techniques that support development and deployment, and external dependencies of the software. The practices of infrastructuring become an explicit part of software development, either in the form of retrospectives and joint explorations of new tools, or as tasks of software engineers and teams whose responsibility it is to support the development. The adoption of CSE, therefore, is never 'finished'. Rather, the change itself becomes a 'never ending story', where developers continuously adapt their practices to changing conditions and needs. This challenges the understanding of CSE as a specific set of technologies and practices that can be implemented; CSE should be seen as a new way of making use of a toolbox with (changing) technologies, techniques, and practices.

Our results open for research on the specificities of CSE practices that helps to better understand how different practices can be adapted to support CSE in specific contexts, and in this way provide knowledge supporting the continuous improvement of CSE. Here, our analysis proposes that infrastructure and infrastructuring are useful concepts to understand the tight relationship between social and technical structures in modern software development.

CRediT authorship contribution statement

Jacob Nørbjerg: Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing, Conceptualization, Data curation, Formal analysis. **Yvonne Dittrich:** Conceptualization, Formal analysis, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that has been used is confidential.

Acknowledgments

We want to thank the two colleagues who took part in the first part of this research but decided to leave the joint work to pursue other projects.

Appendix A. CSE interview guide

Background and context

Company background

- Product(s)
- Organizational structure

Personal background

- Education
- Time in company. Career
- Present role/function

IT Department

- Size
- Organizational position
- Structure

IT projects

- Purpose/content
- Organization
- For all of the above
- Changes in the past few years (timeframe dependent on the timeframe of a CSE change project)
- Differences across the organization (does everybody do it the same way?)

CoDe background, goals, journey

- What does you/the company mean by CSE?
- Who made the decision to move to CSE? When? Why?

Software development practices and challenges – general

Opening question:
Using a current project (or product or activity), which you are involved with as an example please tell us about

- Your role/function
- Project start/end date
- Purpose
- Organization (members, roles)
- Management
- Planning, follow-up and improvements
- Key activities
- Key tools
- Delivery cycles
- Key challenges, set-backs, issues
- In what ways (if any) is this project different from

[The purpose of the above question is to get a detailed as possible idea about processes, practices and challenges in the company, as well as on their 'maturity' with regard to CoDe.

The questions are general in order to enable us to capture the diversity in maturity, practices, and challenges across the case companies.]

Follow-up questions about the continuous sub-processes*

For each of the continuous* activities in (Fitzgerald and Stol 2015 (precise questions will vary with area and CSE maturity):

- How do you do (continuous) x?
  - Plan?
  - Assign; e.g.; development tasks?
  - Challenges/issues
- How do you experiment, monitor and (continuously) improve x?
- How did you move/are you moving towards (continuous) x?
  - Has it changed since the beginning of the continuous journey?
  - Who initiated the change (top-down? Bottom up?)?
  - How did it diffuse into the organization?
  - Challenges/barriers?
- How do you perceive the connection/interaction between the business, development and operations?
  - Could it be better? In what ways? What would it require to improve the relationship/interaction?
- What would you like to see done differently? Why?

Appendix B. Coding scheme

Table B1

Initial code	Aggregated codes[1, 2]	Axial codes (section number)
Bottom Up	*Introduction of CSE*	Transition to CSE (4.1)
Top Down		
Challenges		
Enabling condition		
Kind of Software	*Company*	
Business Model		
Business Context		
Control over Deployment		
Effects of CSE	*Effects of CSE*	
Motivation for CSE	*Motivation for CSE*	
Business Interaction	*Business and Customer Interaction*	
Customer Interaction		
Customer Trust		
Support		
Feedback from Use		
User/customer testing		
Design Heuristics	*Architecture Design*	Product Architecture (4,2)
Microservices		
API		
Feature Toggle		
Database		

Table B.1
Coding scheme.

Initial code	Aggregated codes[1, 2]	Axial codes (section number)
Tooling	*Infrastructure*	Development and Deployment technologes (4.3)
Development of Infrastructure		
Infrastructure for Development		
Infrastructure for Deployment		
Cloud		
1-Button Deployment		
Continuous Integration		
Configuration Management		
Release manager	*Release process*	
Release strategy		
Staging		
Deployment Frequency		
3rd party process dependencies		
Product Owner	*Planning*	Planning and management (4.4)
Long term Planning		
Development planning		
Task Allocation		
Prioritisation of tasks		
Grooming		
Apportioning of changes		
Roles	*Dev Organisation*	
Team organization		
Initial code	**Aggregated codes[1, 2]**	**Axial codes (**section **number)**
Intra Team Coordination	*Coordination*	Coordination and collaboration (4.5)
Dev coordination		
Inter-Team Coordination		
Internal Open Source		
Awareness		
Knowledge Sharing		
Communication		
Code ownership		
Communication tool-dev		
Skills	*Human Aspects*	
Management Mindset		
Developer Mindset		
Responsibility		
Trust (Team)		
Developer Motivation		

Note:.
(1) An initial aggregated code 'Improvement' turned out to be related to all other dimensions and became the topic of the article.
(2) Some aggregated codes had only one initial code.

References

Agarwal, R., Tiwana, A., 2015. Editorial—evolvable systems: through the looking glass of IS. Inf. Syst. Res. 26 (3), 473–479. https://doi.org/10.1287/isre.2015.0595.

Beck, K., Beedle, M., Bennekum, A.v., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D, 2001. Manifesto For Agile Software Development. from agilemanifesto.org.

Bellomo, S., Ernst, N., Nord, R., Kazman, R., 2014. Toward design decisions to enable deployability: empirical study of three projects reaching for the continuous delivery holy grail. In: Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, pp. 702–707.

Button, G., Sharrock, W., 1994. Occassioned practices in the work of software engineers. Requirements Engineering. Academic Press, pp. 217–240.

Claps, G.G., Berntsson Svensson, R., Aurum, A., 2015. On the journey to continuous deployment: technical and social challenges along the way. In: , pp. 21–31. https://doi.org/10.1016/j.infsof.2014.07.009.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46. https://doi.org/10.1177/001316446002000104.

Conboy, K., 2009. Agility from first principles: reconstructing the concept of agility in information systems development. Inf. Syst. Res. 20 (3), 329–354. https://doi.org/10.1287/isre.1090.0236.

Cresswell, J.W., Cresswell, J.D., 2018. Research Design. Qualitative, Quantitative, and Mixed Methods Approaches, 5 ed. Los Angeles, SAGE.

Dennehy, D., Conboy, K., 2016. Going with the flow: an activity theory analysis of flow techniques in software development. J. Syst. Softw. (133), 160–173. https://doi.org/10.1016/j.jss.2016.10.003.

Dittrich, Y., 2014. Software engineering beyond the project – Sustaining software ecosystems. Inf. Softw. Technol. 56 (11), 1436–1456. https://doi.org/10.1016/j.infsof.2014.02.012.

Dittrich, Y., 2016. What does it mean to use a method? Towards a practice theory for software engineering. In: , pp. 220–231. https://doi.org/10.1016/j.infsof.2015.07.001.

Dittrich, Y., Eriksén, S., Hansson, C., 2002. PD in the wild; evolving practices of design in use. In: Proceedings of the PDC. ACM, pp. 23–25.

Dittrich, Y., Michelsen, C.B., Tell, P., Lous, P., Ebdrup, A., 2020. Exploring the evolution of software practices. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ACM, pp. 493–504.

Draxler, S., Stevens, G., 2011. Supporting the collaborative appropriation of an open software ecosystem. Comput. Support. Cooper. Work (CSCW) 20 (4), 403–448.

Draxler, S., Stevens, G., Boden, A., 2014. Keeping the development environment up to date. A study of the situated practices of appropriating the eclipse IDE. IEEE Trans. Softw. Eng. 40 (11), 1061–1074.

Elazhary, O., Storey, M.A., Ernst, N.A., Paradis, E., 2021. ADEPT: a socio-technical theory of continuous integration. In: Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), pp. 26–30. https://doi.org/10.1109/icse-nier52604.2021.00014.

Elazhary, O., Werner, C., Li, Z.S., Lowlind, D., Ernst, N.A., Storey, M.A., 2022. Uncovering the benefits and challenges of continuous integration practices. Trans. Softw. Eng. 48 (7), 2570–2583. https://doi.org/10.1109/TSE.2021.3064953.

Fitzgerald, B., Russo, N.L., Stolterman, E., 2002. Information systems development. Methods in Action, 1 ed. McGraw-Hill.

Fitzgerald, B., Stol, K.J., 2015. Continuous software engineering: a roadmap and agenda. J. Syst. Softw. (123), 1–14. https://doi.org/10.1016/j.jss.2015.06.063.

Giraldo-Mora, J.C., 2023. It is Along Ways. Global Payment Infrastructure in Movement. Monograph, Copenhagen Business School, Copenhagen.

Guckenheimer, S., 2015. Our Journey to Cloud Cadence, Lessons Learned At Microsoft Developer Division. Microsoft Corporation.

Hemon, A., Lyonnet, B., Rowe, F., Fitzgerald, B., 2020. From Agile to DevOps: smart Skills and Collaborations. Inf. Syst. Front. 22 (4), 927–945. https://doi.org/10.1007/s10796-019-09905-1.

Henfridsson, O., Bygstad, B., 2013. The generative mechanisms of digital infrastructure evolution. MIS Q. 37 (3), 907–931. http://www.jstor.org/stable/43826006.

Karasti, H., 2014. Infrastructuring in participatory design. In: Proceedings of the 13th Participatory Design Conference, pp. 141–150.

Karasti, H., Blomberg, J., 2018. Studying Infrastructuring Ethnographically. Comput. Support. Coop. Work (CSCW) 27 (2), 233–265. https://doi.org/10.1007/s10606-017-9296-7.

Karasti, H., Syrjänen, A.L., 2004. Artful infrastructuring in two cases of community PD. In: Proceedings of the Eighth Conference on Participatory Design: Artful Integration: Interweaving Media, Materials and Practices, pp. 20–30.

Klotins, E., Gorschek, T., 2022. Continuous software engineering in the wild. In: Proceedings of the International Conference on Software Quality. Cham. Springer International Publishing, pp. 3–12.

Klotins, E., Gorschek, T., Sundelin, K., Falk, E., 2022. Towards cost-benefit evaluation for continuous software engineering activities. Empir. Softw. Eng. 27 (6) https://doi.org/10.1007/s10664-022-10191-w.

Klotins, E., Peretz-Andersson, E., 2022. The unified perspective of digital transformation and continuous software engineering. In: Proceedings of the 5th International Workshop on Software-intensive Business: Towards Sustainable Software Business, pp. 75–82. https://doi.org/10.1145/3524614.3528626.

Klotins, E., Talbert-Goldstein, E., 2023. Organizational conflicts in the adoption of continuous software engineering. In: Proceedings of the International Conference on Agile Software Development. Springer, pp. 149–164.

Knorr-Cetina, K., 2001. Objectual practice. In: Schatzki, T.R., Cetina, K.K., Savigny, E.v. (Eds.), The Practice Turn in Contemporary Theory. London, Routledge, pp. 184–197.

Laukkanen, E., Itkonen, J., Lassenius, C., 2017. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. In: , pp. 55–79.

Leite, L., Pinto, G., Meirelles, P., 2021. The organization of software teams in the quest for continuous delivery: a grounded theory approach. In: .

Leppänen, M., Mäkinen, S., Pagels, M., Elorante, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T., 2015. The highways and country roads to continuous deployment. In: *Proceedings of the IEEE Software*:March/April), pp. 64–72.

Lous, P., Tell, P., Michelsen, C.B., Dittrich, Y., Ebdrup, A., 2018a. From Scrum to Agile: a journey to tackle the challenges of distributed development in an Agile team. In: Proceedings of the International Conference on Software and System Process. Chicago, pp. 11–20.

Lous, P., Tell, P., Michelsen, C.B., Dittrich, Y., Kuhrmann, M., Ebdrup, A., 2018b. Virtual by design: how a work environment can support agile distributed software development. In: Proceedings of the IEEE/ACM 13th International Conference on Global Software Engineering (ICGSE). IEEE, pp. 97–106.

Lwakatare, L.E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H.H., Bosch, J., Oivo, M., 2016. Towards DevOps in the embedded systems domain: why is it so hard?. In: Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS), p. 2016. https://doi.org/10.1109/HICSS.2016.671, 5-8 Jan5437-5446.

Metz, C., 2016. Here's How Google Makes Sure It (Almost) Never Goes Down WIRED. Wired: Wired.

Neely, S., Stolt, S., 2013. Continuous delivery? Easy! just change everything (Well, maybe it is not that easy). In: Proceedings of the Agile Conference (AGILE), pp. 121–128. https://doi.org/10.1109/AGILE.2013.17. *2013*.

O'Connor, R.V., Elger, P., Clarke, P.M., 2017. Continuous software engineering-a microservices architecture perspective. J. Softw. Evol. Process 29 (11), e1866. https://doi.org/10.1002/smr.1866.

Osmundsen, K., Bygstad, B., 2021. Making sense of continuous development of digital infrastructures. J. Inf. Technol. 37 (2), 144–164. https://doi.org/10.1177/02683962211046621.

Pipek, V., Karasti, H., Bowker, G.C., 2017. A preface to 'infrastructuring and collaborative design. In: Proceedings of the Computer Supported Cooperative Work (CSCW) 26. Springer, pp. 1–5.

Pipek, V., Wulf, V., 2009. Infrastructuring: toward an integrated perspective on the design and use of information technology. J. Assoc. Inf. Syst. 10 (5) https://doi.org/10.17705/1jais.00195.

Robson, C., McCartan, K., 2016. Real World Research: A Resource For Users of Social Research Methods in Applied Settings. Wiley.

Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., Stumm, M., 2016. Continuous deployment at Facebook and OANDA. In: Proceedings of the 38th International Conference on Software Engineering, pp. 21–30. https://doi.org/10.1145/2889160.2889223.

Schmidt, K., Simone, C., 1996. Coordination mechanisms: towards a conceptual foundation of CSCW systems design. Comput. Support. Coop. Work 5 (2–3), 155–200. *The Journal of Collaborative Computing*.

Shahin, M., Babar, M.A., Zhu, L., 2017. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access, pp. 3903–3943. https://doi.org/10.1109/ACCESS.2017.2685629.

Shahin, M., Zahedi, M., Babar, M.A., Zhu, L., 2018. An empirical study of architecting for continuous delivery and deployment. Empir. Softw. Eng. 24 (3), 1061–1108. https://doi.org/10.1007/s10664-018-9651-4.

Sigfridsson, A., 2010. The Purposeful Adaptation of Practice: an Empirical Study of Distributed Software Development. University of Limerick, Limerick.

Sigfridsson, A., Avram, G., Sheehan, A., Sullivan, D.K., 2007. Sprint-driven development: working, learning and the process of enculturation in the PyPy community. In: Proceedings of the IFIP International Conference on Open Source Systems. Springer, pp. 133–146.

Star, S.L., and Bowker, G.C., (2006). How to infrastructure. In: *Handbook of new media: Social shaping and social consequences of ICTs*, pp. 230–245.

Star, S.L., Ruhleder, K., 1996. Steps toward an ecology of infrastructure: design and access for large information spaces. Inf. Syst. Res. (7:1), 111–134.