Aleksi Lokka

CONTINUOUS DELIVERY ADOPTION CHALLENGES FOR SMALL AND MEDIUM SIZED ERP SYSTEM VENDORS

UNIVERSITY OF JYVÄSKYLÄ
FACULTY OF INFORMATION TECHNOLOGY
2020

ABSTRACT

Lokka, Aleksi
Continuous Delivery adoption challenges for small and medium sized ERP system vendors
Jyväskylä: University of Jyväskylä, 2020, 69 pp.
Information systems, Master's Thesis
Supervisor: Seppänen, Ville

As the business environments and the requirements of enterprise software users are evolving increasingly faster, enterprise system vendor organizations have to develop their products increasingly faster, while maintaining a high level of software quality. To address these demands, software companies have in the recent decade adopted a software engineering practice known as Continuous Delivery (CD), in which the software is maintained in a releasable state, but not deployed necessarily automatically. However, there are certain application domains that do not facilitate the adoption of Continuous Delivery, such as Enterprise Resource Planning (ERP) systems, as they are mission-critical, large, and complex software systems designed to manage all central business functions of an organization, yet the ERP system vendors are facing the same demands as other software vendors and are therefore seeking to adopt the Continuous Delivery practice in their development activities.

The objective of this study is to identify and analyze the challenges related to adoption of Continuous Delivery practice in small to medium sized Enterprise Resource Planning (ERP) system vendors. The study is divided into two sections: a literature review of previous research, and an empirical qualitative study, in which five industry professionals representing four organizations were interviewed. As a result, a framework consisting of 12 identified Continuous Delivery adoption challenges classified into five separate, but interconnected challenge themes is presented.

Keywords: software engineering, continuous delivery, enterprise resource planning

TIIVISTELMÄ

Lokka, Aleksi
Jatkuvan toimittamisen käyttöönoton haasteet pienille ja keskisuurille toiminnanohjausjärjestelmien toimittajille
Jyväskylä: University of Jyväskylä, 2020, 69 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Seppänen, Ville

Liiketoiminnan ympäristöjen ja yritysjärjestelmien käyttäjien vaatimusten kehittyessä jatkuvasti kiihtyvällä tahdilla yritysjärjestelmien toimittajien on kehitettävä tuotteitaan yhä nopeammin, samalla säilyttäen ohjelmistojen korkean laadun. Vastatakseen näihin vaatimuksiin, ohjelmistoyritykset ovat viimeisen vuosikymmenen aikana ottaneet käyttöön jatkuvana toimittamisena tunnetun ohjelmistotuotannon menetelmän, jossa ohjelmisto pidetään jatkuvasti julkaisukelpoisena. Tietyt ohjelmistotyypit, kuten toiminnanohjausjärjestelmät, joilla yritykset hallinnoivat kaikkia keskeisiä liiketoiminnan osa-alueitaan, eivät kuitenkaan erityisesti sovellu jatkuvasti toimitettaviksi niiden toiminnan kriittisyyden, laajuuden, tai monimutkaisuuden vuoksi. Samat vaatimukset koskettavat kuitenkin toiminnanohjausjärjestelmien toimittajia kuin muitakin ohjelmistotoimittajia, minkä vuoksi myös toiminnanohjausjärjestelmien toimittajat pyrkivät liittämään jatkuvan toimittamisen menetelmiä osaksi ohjelmistokehitystään.

Tämän tutkimuksen tavoitteena on tunnistaa ja analysoida pienten ja keskisuurien toiminnanohjausjärjestelmien toimittajien haasteita jatkuvan toimittamisen käyttöönottoon liittyen. Tutkimus on jaettu kahteen osaan: kirjallisuuskatsaukseen ja empiiriseen laadulliseen tutkimukseen, jossa haastateltiin viittä alan asiantuntijaa, jotka edustivat neljää eri järjestelmätoimittajaa. Tutkimuksen tuloksena esitellään viitekehys, joka sisältää yhteensä 12 jatkuvan toimittamisen käyttöönoton haastetta luokiteltuna viiteen erilliseen, mutta toisiinsa liittyvään teemaan.

Asiasanat: ohjelmistotuotanto, jatkuva toimitus, toiminnanohjausjärjestelmä

FIGURES

TABLES

TABLE OF CONTENTS

1 INTRODUCTION

The modern cizilization is increasingly dependent on digital technologies, products, and services. This is true for not only individuals, but also for business organizations, which will have to adapt to the constantly evolving business environment, which has changed increasingly faster since the introduction of Information Technology and different types of software systems that are used to enhance the performance of business organizations. A prominent class of enterprise software systems referred to as Enterprise Resource Planning (ERP) systems have been used for decades to manage most of the central business functions of businesses. As the systems have evolved over time, so have the requirements for the systems set by their users. Increasingly changing business environments require increasingly evolving software solutions, and ERP systems are no exception. As a result, the ERP system vendors have to be able to deliver their systems more efficiently, flexibly, and rapidly in order to accommodate the differing and changing needs of their customers.

One possible solution to address the problem for system vendors is to adopt a software engineering method known as Continuous Delivery, in which the software is constantly kept in a releasable state, and upgrades such as additional system features can be delivered to the users at any time. The proposed benefits of Continuous Delivery include shorter time-to-market and improved software quality, customer satisfaction, and reliability of software releases. Adopting the Continuous Delivery practice, however, is remarkably difficult due to the critical characteristics of ERP systems, and other software development challenges in the ERP system vendor organizations.

The objective of this study is to identify the challenges related to adoption of Continuous Delivery practice in small to medium sized ERP system vendors. In order to identify and analyze the challenges, the following research question was formulated:

- *What are the Continuous Delivery adoption challenges for small and medium sized ERP system vendors*?

In order to address the research problem, the study was divided into two sections: a literature review and empirical research. The objective of the literature review was to obtain understanding of the phenomenon of interest, including essential terminology and central concepts. The literature review was conducted by searching for research articles published in scientific publications, such as journals and conference papers, with prioritization of research articles to be included in the literature review as the following, in order: relevancy of the research article, quality of the publication, and time of the publication. The relevancy of the research article refers to the level of similarity between the topic of the article and the research problem. The quality of the publication was determined by the amount of references of the article and the overall recognizability of the publication, which was partly evaluated by utilizing a publication classification system Julkaisufoorumi. Finally, the time of publication was considered as more recent research articles were estimated to produce more relevant data to address the research problem.

The research articles for the literature review were initially searched from the eight leading Information Systems journals, known as Senior Scholars' Basket of Journals (Association for Information Systems, 2011), which are the following, in alphabetical order: European Journal of Information Systems (EJIS), Information Systems Journal (ISJ), Information Systems Research (ISR), Journal of AIS (JAIS), Journal of Information Technology (JIT), Journal of MIS (JMIS), Journal of Strategic Information Systems, and MIS Quarterly (MISQ). In addition, relevant research databases such as AIS eLibrary, ACM Digital Library, and IEEE Xplore were employed in order to find relevant research for the literature review. Finally, Google Scholar search engine was also used to obtain further coverage of relevant research articles. As the existing research on some of the key concepts proved to be extremely limited, if not nonexistent, practitioner white papers, articles, and blog entries were used as substitutes for research articles, but only when absolutely necessary or otherwise unavoidable. The keywords that were used to obtain relevant research articles were the following: "enterprise resource planning", "erp", "erp ii", "erp iii", "continuous delivery", "cd", "continuous integration", "ci", "continuous software delivery", "release management", "deployment pipeline", and combinations of the previous keywords, e.g. "erp continuous delivery", "continuous delivery release management", and "continuous delivery deployment pipeline". Furthermore, as the research problem suggests, words "problem" and "challenge" were also combined with previous keywords, e.g. "continuous delivery challenge".

The main objective of the empirical study was to examine what challenges ERP system vendors face when adopting and practicing continuous software delivery methods. As the objective of this study is to describe and analyze a phenomenon instead of explaining it or making predictions related to it, the manifesting body of knowledge, or theory, can be considered analytic in nature (Gregor, 2006). As the Continuous Delivery of software has been a subject of extensive research for over a decade, it is no surprise that there are multiple extensive systematic literature reviews regarding the challenges of adopting them published recently, such as research articles by Laukkanen et al. (2017a) and Shahin et al. (2017). The research on how Continuous Delivery practices are adopted and applied in the explicit domain of ERP systems development, however, appears to be extremely limited, if not nonexistent, especially from the viewpoint of ERP system vendors instead of the user organizations. Because of that, a clear research gap exists and thus this study provides a scientific contribution to the academic discipline of Information Systems.

The rest of this thesis is structured as follows: in the chapters two and three, the essential concepts of Continuous software delivery and Enterprise Resource planning, respectively, are discussed on the foundation of the conducted literature review. In chapter four, the research methodology of the empirical research is discussed, including the research, data collection and analysis methods. In chapter five, the results of the empirical research are presented. In chapter six, the results are analyzed and discussed, and a novel theory in a form of conceptual framework is proposed. In chapter seven, the theoretical and practical implications and limitations are discussed, and topics for further research are considered. Last, in chapter eight, a conclusion of the study is presented.

2 CONTINUOUS SOFTWARE DELIVERY

Continuous delivery of software is increasingly more prominent practice among the software industry. It can be seen as an extension of agile methods, which in turn were developed to overcome challenges and issues caused by traditional, life-cycle based software development and engineering methods, such as the waterfall method. In this chapter, the history of software engineering and agile development methods are summarized, and the practices of Continuous Integration, deployment pipeline, and Continuous Delivery are discussed in detail with their proposed benefits and challenges also summarized. In addition, related concepts of Continuous Deployment and DevOps, are discussed.

2.1 Software engineering

The term software engineering, first introduced in 1968 (Wirth, 2008), means "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software" (IEEE, 1990, p. 67). One of the fundamental elements of traditional software engineering is the concept of software development life cycle, which is essentially a tool to model how software is designed, created, and maintained (Davis, Bersoff & Comer, 1988). Arguably the most well-known life cycle model is a sequential model first depicted by Royce (1970), later named the waterfall model, in which several phases of development activity have to be completed in strict order for software to be deliverable. The original waterfall model consisting of six sequential phases is depicted in the Figure 1 (Royce, 1970).

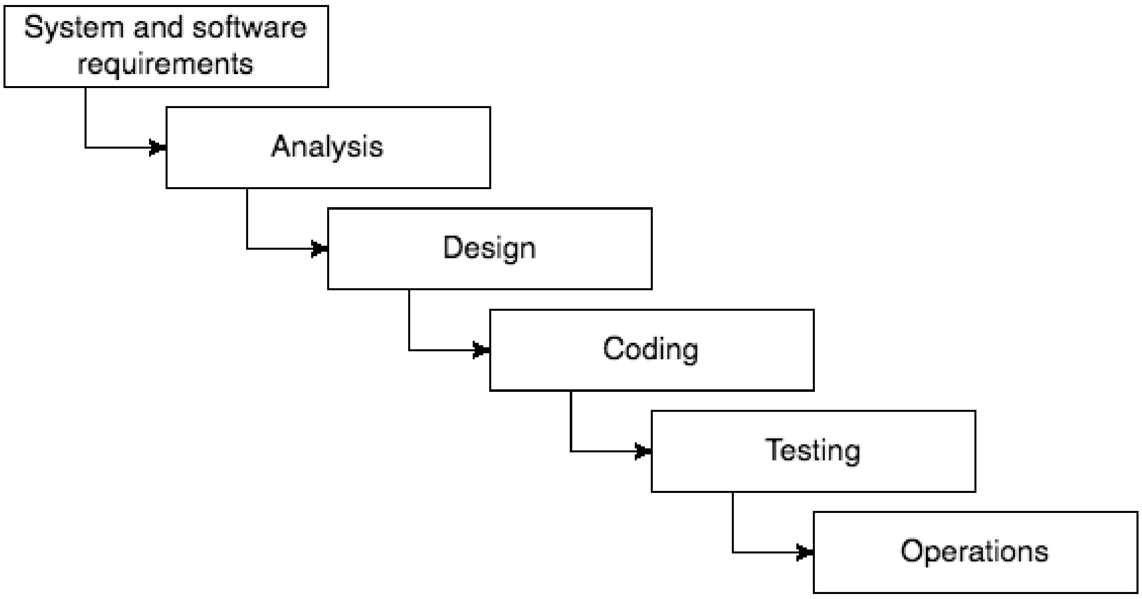

FIGURE 1 Waterfall development model (Royce, 1970)

The sequential software development process models, including the waterfall model, have received criticism for a variety of reasons. For example, such models do not provide feedback between the phases, and because of the sequential nature problems with specification, design, and implementation are discovered only when the system has been integrated. Furthermore, once a specification has been locked, it is difficult to change in response to changing user needs, even though both organizational and end-user requirements are known to change during the development process, thus causing a constant pressure for respecification. As a result, the delivered software may not meet the actual requirements of the customer. (Sommerville, 1996.)

2.2 Agile methods

First instances of agile methods, or development methods that can be considered as such in retrospect, were developed in the 1970s, or even before. One of these methods that influenced the emergence of agile software development methods is the "iterative enhancement" technique that was introduced in 1975 (Basili & Turner, 1975). These methods, however, did not succeed to achieve popularity. It was not until the mid to late 1990s, when new development methods, such as Extreme Programming (XP), were developed, that agile methods finally started to reach mainstream success in the software industry (Cohen, Lindvall & Costa, 2003). A defining moment in the history of software engineering occurred in 2001, when a group of software professionals published the "Agile Software Development Manifesto" (Fowler & Highsmith, 2001). The four foundational values of the Agile Manifesto are as follows:

- Individuals and interactions over processes and tools
- Working software over compehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Individuals and interactions over processes means that the relationships and communality of the software developers and their roles are emphasized instead of institutionalized processes and development tools. This is manifested in form of procedures improving team spirit, such as close team relationships, and working environment arrangements. (Abrahamsson, Salo, Ronkainen & Warsta, 2002.) *Working software over comprehensive documentation* implies that the most important objective of a software development team is to continuously produce tested and working software. This is achieved by producing new releases frequently, in intervals from hourly to daily releases to monthly or bi-monthly. The developers should keep the code simple and technically superb in order to avoid unnecessary and burdening documentation. (Abrahamsson et al., 2002.)

Customer collaboration over contract negotiation means that the relationship and collaboration between the developers and the customers is preferred over rigid contracts. However, the larger a software project is, the more important it is to have a well-drafted and negotiated contract. The objective of negotiations is to be able to form a lasting relationship. Agile development emphasizes the delivery of business value as soon as the project begins, reducing the risks of contract breach in form of a non-fulfillment. (Abrahamsson et al., 2002.) *Responding to change over following a plan* refers to the endorsed practice of both software developers and customer representatives to be well informed, competent, and authorized to address the needs that may emerge during the development process. Each of the participants should be able to make changes, and the contracts should be made in a way to make that possible. (Abrahamsson et al., 2002.)

While each of the values discussed above is meaningful, arguably the most important precedent for latter developments in the software engineering discipline is the second one, delivering working software over comprehensive documentation. According to Abrahamsson et al. (2002), the "working software over comprehensive documentation" implies that the software development teams should continuously deliver tested and working software, with releases occurring frequently, even hourly or daily. Agile methods contrast the traditional software development approaches that are guided by a SDLC model, including the waterfall model, which emphasize process-centricity, specified tasks and their desired outcomes, specialized roles, extensive documentation, and formalized communication through that documentation (Nerur, Mahapatra & Mangalaraj, 2005). The main differences between traditional and agile software development are summarized in Table 1.

TABLE 1 Traditional vs. agile software development (Nerur et al., 2005, p. 75)

	Traditional	**Agile**
Fundamental Assumptions	Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning.	High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change.
Control	Process centric	People centric
Management Style	Command-and-control	Leadership-and-collaboration
Knowledge Management	Explicit	Tacit
Role Assignment	Individual—favors specialization	Self-organizing teams—encourages role interchangeability
Communication	Formal	Informal
Customer's Role	Important	Critical
Project Cycle	Guided by tasks or activities	Guided by product features
Development Model	Life cycle model (Waterfall, Spiral, or some variation)	The evolutionary-delivery model
Desired Oganizational Form/Structure	Mechanistic (bureaucratic with high formalization)	Organic (flexible and participative encouraging cooperative social action)
Technology	No restriction	Favors object-oriented technology

2.3 Continuous Integration

Continuous integration (CI) is a software development and engineering practice where the software code created by a development team is integrated frequently, usually multiple times per day. Each integration is built and tested automatically with a purpose to detect mistakes and other errors in the integration as early as possible. (Fowler, 2006.) This contrasts the formerly common practice of integrating the work of individual software developers and development teams only after longer passages of development work, ranging from multiple days to even months, which can lead to increase to possibility and severity of integration conflicts between lines of work to increase (Laukkanen, Itkonen & Lassenius, 2017). Thus, one of the main rationales behind Continuous Integration practice is the fact that the sooner errors and conflicts are detected, the easier and quicker they are to correct. At the same time, difficulties with long-running code branches and merge conflicts related to them are avoided (Meyer, 2014).

New code changes from a version control are usually tracked by a dedicated CI server, which also builds and tests the system automatically after each code change (Meyer, 2014). In summary, efficient Continuous Integration practices and tools automate the compilation, building, and testing phases of software development (Hilton, Nelson, Tunnell, Marinov & Dig, 2017).

Although similar practices have probably been used earlier, the term and concept of Continuous Integration first appeared in the late 1990s as one of the twelve basic practices in the Extreme Programming (XP) software development methodology (Ståhl & Bosch, 2014). In the XP, the practice of Continuous Integration proposes that new code should be integrated with the current system within only a few hours, and each time the integration takes place, the entire system should be built from scratch. If the build does not pass each and all of the tests, all changes are discarded (Beck, 1999). For the most part, the current practice of Continuous Integration as a part of Continuous Delivery is quite similar. However, the exact definition of Continuous Integration is still somewhat unclear, as there "is currently no consensus on Continuous Integration as a single, homogenous practice" (Ståhl & Bosch, 2014, p. 57.) The following benefits of Continuous Integration practice have been reported (Fowler, 2006; Lehtonen, Suonsyrjä, Kilamo & Mikkonen, 2015; Ståhl & Bosch, 2014):

- Reduced risk
- Faults are detected earlier and are easier to track
- Improved code quality
- More frequent deployments
- Increased developer productivity

2.4 Deployment pipeline

Deployment pipeline, also known as Continuous Integration pipeline (Humble & Farley, 2010; Hilton et al., 2017), consists of several subsequent stages with predetermined requirements that software build must fulfill to pass through in the pipeline in order to be releasable for the end users, or into the production environment (Humble & Farley, 2010; Steffens, Lichter & Döring, 2018). The deployment pipeline is different for each software application with possible additional stages included, but in general, it consists, at least of the following four stages: commit stage, automated acceptance test stage, manual test stage, and release stage, which are depicted in the Figure 2 (Humble and Farley, 2010).

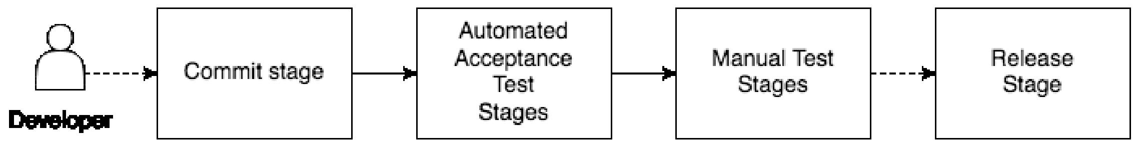

FIGURE 2 Outline of the deployment pipeline stages (Humble & Farley, 2010)

The commit stage affirms that the system functions at the technical level. The code is compiled and it will pass a sequence of primarily unit-level automated tests, after which a code analysis is performed (Humble & Farley, 2010). *The automated acceptance test stages* affirm that the system works not only at the functional level, but also at the nonfunctional level. In addition, it is assured that the system meets the behavioural requirements of its users and the specifications of the client. (Humble & Farley, 2010.) The objective of *the manual test stages* is to affirm that the system is working as intended and fulfills its requirements, to expose defects not detected by automated tests, and to verify that the system provides value to its users. The manual test stages can usually include environments of exploratory testing and integration, and user acceptance testing as well. (Humble & Farley, 2010.) Finally, in *the release stage*, the system is delivered to its users, either as packaged software or as a deployment into a production environment or similar, namely a testing environment that is identical to the production environment. (Humble & Farley, 2010.)

Fundamentally, the deployment pipeline is an automated process of delivering software, that is most often manifested in a form of integrated software system, which utilizes changes in source code retrieved from the version control system as an input, and functional software releases as an output, which can then be deployed into the production environment. Furthermore, the deployment pipeline provides immediate feedback to each of stakeholders involved at every stage of the software delivery process. (Lehtonen et al., 2015.) Similarly as the deployment pipeline is an extension of the practice of Continuous Integration, a functional deployment pipeline is a prerequisite for practice of Continuous Delivery (Lehtonen et al., 2015; Olsson et al., 2012).

The terminology of the deployment pipeline has faced criticism, as the deployment pipeline can be seen either as a model, software system, or process. Some researches, such as Steffens et al. (2018), have criticized this multidimensional definition of deployment pipeline for its ambiguity, and have proposed more consistent terminology instead: a deployment pipeline should be disclosed as a software delivery model, software delivery system, or software delivery process, depending on which one of the previously mentioned dimensions is in question. However, as this classification has not reached any significant status in the research, it is discarded in context of this thesis and the more customary term deployment pipeline is used when referring to any of the three previously mentioned dimensions.

2.5 Continuous Delivery

Continuous Delivery (CD) is "a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time" (Chen, 2015; p. 50). By another definition, Continuous Delivery is a set of principles, patterns, and practices that enable software deployments to become routine tasks that can be performed on demand at any time (Humble, 2018). Indeed, while there is no formal definition for Continuous Delivery, "there appears to be some level of agreement amongst authors that Continuous Delivery refers to the ability of an organization to release software functionality directly to customers on demand and at will (deployment), faster and more frequently than traditional software development" (Rodríguez et al., 2017, p. 274). The main rationale for adopting the Continuous Delivery practice is to be able to get all changes in software, regardless of their size and scope, available to the end users in a safe, sustainable, and quick manner (Humble, 2018).

The concept of Continuous Delivery and its relationship between Continuous Integration and Continuous Deployment is illustrated in the Figure 3 (Shahin, Zahedi, Babar Zhu, 2019).

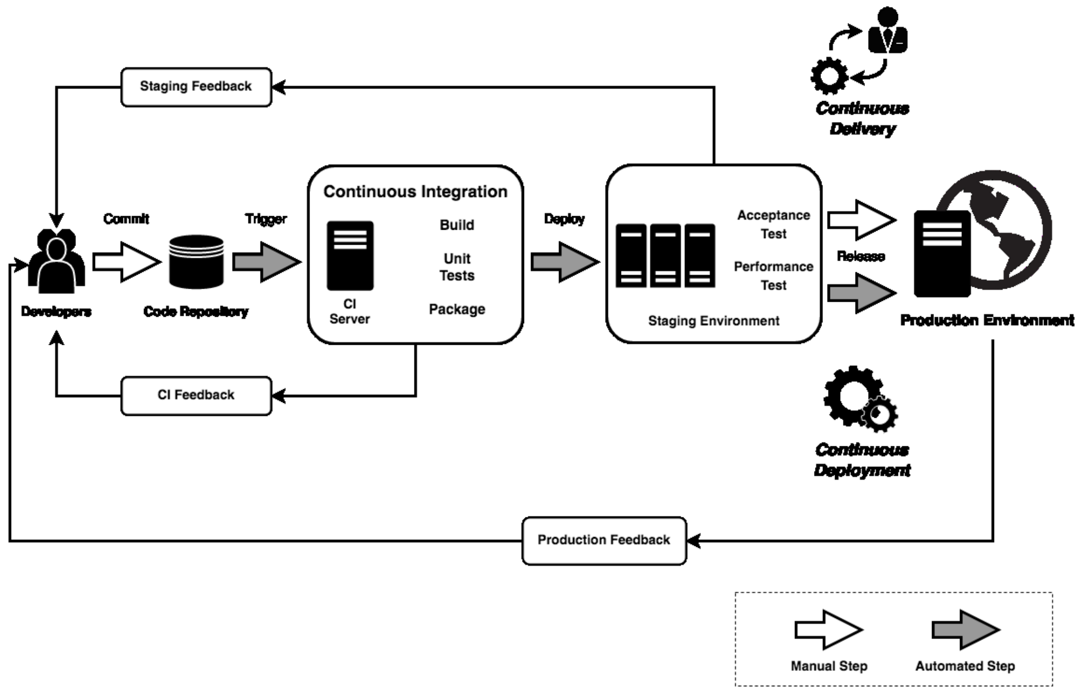

FIGURE 3 The relationship between Continuous Integration, Continuous Delivery, and Continuous Deployment (Shahin et al., 2019)

2.5.1 Benefits

Adopting and practicing Continuous Delivery will provide a software development organization with multiple benefits, some of which are connected to each other. The benefits reported in the research literature include shortened time-to-market and continuous feedback, improved productivity, quality, customer satisfaction, and release reliability. The benefits are presented in Table 2.

TABLE 2 Continuous Delivery benefits

Benefit	Literature

Shorter time-to-market, more frequent releases	Olsson et al., 2012; Neely & Stolt, 2013; Chen, 2015; Leppänen et al., 2015
Improved productivity and software quality	Humble et al. 2006; Chen, 2015; Leppänen et al., 2015; Itkonen et al., 2016
Improved customer satisfaction	Neely & Stolt, 2013; Chen, 2015; Leppänen et al., 2015;
Improved release reliability, decreased risk of release failures	Humble et al., 2006; Neely & Stolt, 2013; Chen, 2015; Itkonen et al., 2016
Faster, continuous feedback	Olsson et al., 2012; Neely & Stolt, 2013; Leppänen et al., 2015

In addition to benefits presented above, authors have also identified benefits such as "building the right product", (Chen, 2015) "effort savings" and "a closer connection between development and operations" (Leppänen et al., 2015). Furthermore, in their case study of a Finnish digital service provider, Itkonen et al. (2016) reported also the following additional benefits of adopting Continuous Delivery:

- Improved collaboration
- Organisational independence
- Infrastructural agnosticism
- Improved developer morale

In summary, a number of benefits are reported and their significance to overall performance of an organisation is remarkable. The benefits are not limited only to improvements in the development processes itself, but also factor in organizational aspects as well.

2.5.2 Challenges and problems

Researchers have pursued to gain understanding and to summarize the challenges and problems related to the adoption and practice of Continuous Delivery in the form of extensive systematic literature reviews and mapping studies. (Laukkanen, Itkonen & Lassenius, 2017; Rodríguez et al., 2017; Shahin, Ali Babar & Zhu, 2017) In their systematic literature review, Laukkanen et al. (2017a) identified a total of 40 different Continuous Delivery problems classified into seven themes, which are the following:

- Build design
- System design
- Integration
- Testing
- Release
- Human and organizational
- Resource

Build design problems are caused by build design decisions, such as having a complicated build system, leading to complex builds, or having a build system that cannot be adjusted, leading to inflexible builds. Complex and inflexible builds are difficult to modify and maintain, and may break more often. (Laukkanen et al., 2017a.) *System design problems* are caused by system design decisions, such as having a system that consists of multiple modules or services, or having an unsuitable architecture for continuous delivery. Neither of them are problems on their own, but the effects they have, as they can lead to increased development effort, testing complexity, and problematic deployment. (Laukkanen et al., 2017a.) *Integration problems* occur when the source code is integrated into the mainline, e.g. commits containing large amounts of changes, merge conflicts, broken builds, blockages of work, long-running branches, and slow approval of integration (Laukkanen et al., 2017a). *Testing problems* are related to software testing. They include problems such as ambiguous test results, randomly failing tests, time-consuming testing, untestable code, problematic deployment, and complex testing. (Laukkanen et al., 2017a.) *Release problems* cause issues when the software is released, such as customer data preservation, feature discovery, user dissatisfaction with frequent updates, and deployment downtime (Claps et al., 2014; Laukkanen et al., 2017a). While *human and organizational problems* do not relate to any specific development activity, they are general issues regarding Continuous Delivery adoption, including problems such as lack of discipline, motivation, and experience (Laukkanen et al., 2017a). Last, *resource problems* are related to the availability of resources for the adoption, e.g. required effort, sufficient hardware resources, and network latencies (Laukkanen et al., 2017a).

In turn, Shahin et al. (2017) grouped the Continuous Delivery adoption challenges into six categories, of which two are exclusively for Continuous Delivery adoption, while other four categories of challenges are mutual for Continuous Integration and Continuous Deployment also, with a total of 13 challenges identified. The categories and corresponding challenges are presented in Table 3 (Shahin et al., 2017).

TABLE 3 Continuous Delivery adoption challenges, adapted from Shahin et al. (2017)

	Category	Challenges
Continuous practices adoption challenges	**Team Awareness and Communication**	Lack of awareness and transparency
		Coordination and collaboration challenges
	Lack of investment	Cost
		Lack of experience and skill
		More pressure and workload for team members
		Lack of suitable tools and technologies
	Change resistance	General resistance to change
		Scepticism and distrust on continuous practices
	Organizational processes, structure and policies	Difficulty to change established organizational policies and cultures
		Distributed organization
Challenges exclusive to Continuous Delivery practice	**Lack of suitable architecture**	Dependencies in design and code
		Database schema changes
	Team dependencies	

2.6 Related concepts

In this chapter a few concepts that are related to the continuous software delivery, but not necessarily integral to the research problem, are discussed. These concepts include Continuous Deployment, DevOps, and release planning.

2.6.1 Continuous Deployment

The lack of established software engineering terminology has led to a situation where certain terms are being used in multiple ways, which can cause disorientation among researchers and practitioners. For example, Continuous Delivery and Continuous Deployment are sometimes used interchangeably, even though they are two distinct, yet related, concepts, and should be treated as such. Essentially, Continuous Deployment is a further stage of Continuous Delivery, as in it software is automatically deployed as it is built and tested. In a most extreme situation software would be automatically deployed and released to the end users multiple times a day. In a sense, the main difference between Continuous Delivery and Deployment is whether the software is released to the end users automatically or not (Lehtonen, Suonsyrjä, Kilamo & Mikkonen, 2015). In the former, software is kept releasable but not necessarily released, while in the latter it is made available in the production environment as soon as software has successfully cleared each of the deployment pipeline stages.

2.6.2 DevOps

As the name suggests, DevOps integrates *Development* (Dev) and *Operations* (Ops) by utilizing automation of development, deployment, and infrastructure monitoring (Ebert, Gallardo, Hernantes & Serrano, 2016). It is, however, more than a mere toolset or bundle of practices and standards for software development, as it requires genuine organizational change and culture shift from distributed and separate teams into collaboration between everyone involved in delivering software in an organization (Rajkumar, Pole, Adige & Mahanta, 2016). Humble and Molesky (2011) define the four main principles for DevOps as following: culture, automation, measurement, and sharing.

DevOps is a relatively new phenomenon and therefore related research remains limited, which results in lack of established terminology. This, in part, has an effect on research on the subject and could also inhibit the adoption of DevOps, as organizations do not have a clear perception of what practices should be implemented and how. (Riungu-Kalliosaari, Mäkinen, Lwakatare, Tiihonen & Männistö, 2016)

The practices of Continuous Integration and Continuous Delivery are tightly connected to DevOps, as they are both incorporated into the very core of one of the main objectives of DevOps, which is to automate the software development process and to deliver software more frequently and with higher quality (Steffens, Lichter & Döring, 2018). In that sense, Continuous Integration and Continuous Delivery can be seen not only as components of DevOps, but also as important prerequisites for successful adoption of DevOps. In that sense, as Continuous Integration and Continuous Delivery are practices that enable DevOps (Waller, Ehmke & Hasselbring, 2015), it is reasonable to suggest that successful adoption of DevOps methodology requires also successful implementation of both practices.

3 ENTERPRISE RESOURCE PLANNING SYSTEMS

In this chapter the definition, history, and current state of Enterprise Resource Planning (ERP) systems are discussed, and the common benefits associated with ERP systems are presented.

3.1 Definition

While there is no explicit and universally accepted definition for ERP (Enterprise Resource Planning) system, it is most commonly defined as a suite of software or software system which is used to manage all essential business functions and processes of an organization, e.g. production, finance, supply chain, and customer relationship management, in real-time through integrated data transactions and shared databases between different business processes and departments. What separates ERP systems from other enterprise systems is that it is designed to cover most, if not all business functions, supports real-time information processing, and utilizes shared data transactions within the whole system. (Umble, Haft & Umble, 2003.)

3.2 History and evolution

The origins of the modern ERP systems can be traced back to the early accounting and inventory management systems first developed in the 1960s. By the beginning of the 1970s, these early software systems had evolved into MRP (Material Requirements Planning) systems. (Elragal & Haddara, 2012.) The main rationale for developing and adopting these systems was to be able to estimate the material requirements for production in a more efficient manner, leading to decreased inventory management costs. Starting from the end of the decade, and especially during the 1980s, MRP systems evolved further into systems referred to as MRP II (Manufacturing Resource Planning) systems, which in addition to production planning functionalities included also support for activities related to finance, order management, distribution, and procurement (Elragal & Haddara, 2012).

In the early 1990s, ERP systems were introduced as an extension to preceding MRP and MRP II systems, providing enhanced functionality that covers not only the whole organization and its departments, but also all key business processes, as opposed to MRP and MRP II systems, which covered only the processes related directly to production and manufacturing activities. The first ERP systems, later identified as ERP I systems, were locally installed and monolithic in their architecture, and being generalized software products, provided little to no support for specific business domains or industries apart from manufacturing, unless heavily customized by the user organization, an activity that is considered both expensive and time-consuming, rendering ERP systems almost impossible to be adopted by small and medium enterprises. For this reason, ERP I systems were used almost exclusively by the largest, multinational companies.

By the end of the decade, vendors had begun to produce their ERP software in a more flexible manner, as the ERP system was reconstructed into a collection or suite of software modules that accessed a shared database, with each module corresponding to a certain business function. This way business organizations could choose to have different combinations of modules and functionalities to be implemented according to their needs. Nevertheless, these systems were considered rather inflexible regardless of the modularization and therefore unsuitable for many smaller companies.

From the beginning of the 2000s, vendors started to include new functional modules for supporting additional business processes, such as Supply Chain Management (SCM) and Customer Relationship Management (CRM) in order to address the evolving requirements of their client organizations. These systems are commonly referred to as ERP II systems, and as of the early 2010s, were the still the most dominant type of ERP system in industrial settings (Clegg & Wann, 2013). During the 2000s also another major advancement towards increased flexibility was achieved, as vendors began to offer solutions enabled by cloud computing, namely remote hosting and access. Still, systems were still principally delivered with a licencing fee model with separate payments for possible additional services such as hosting, support, and maintenance (Clegg & Wan, 2013). The distinctive evolutionary trends of the ERP system are summarized in Table 4 (Clegg & Wan, 2013, p. 1461).

TABLE 4 ERP system trends (Clegg & Wan, 2013, p. 1461)

Key element	**ERP**	**ERP II**	**ERP III**

Role of system	Single organization optimization and integration	Multi-organisation participation with some collaborative commerce potential	Multi-organisation, internet based, with full collaborative commerce functionality
Business scope	Manufacturing and distribution focus, automatic business transactions	Often sector-wide offering upstream and downstream integration	Facilitating cross sectors strategic alliances
Functions addressed	Manufacturing, product data, sales and distribution, finance	Most internal organisational functions supported with some limited supplier and customer integration	All internal functions supported plus core inter-company processes
Processes supported	Internal, hidden, with an intra-company boundary	Externally connected with intra-enterprise (i.e. intercompany) focus	Externally connected, open network to create borderless inter-enterprise/industry-wide focus
Information system architecture	Web-aware	Web-based, componentized, non-proprietary	Web-based communication, service-oriented architecture
	Closed and monolithic	Internally and externally available, often subscribed to by joint ventures	External exchange via open source and cloud computing

3.3 Contemporary ERP systems

In addition to the established functionalities for managing core business activities, the contemporary ERP systems, also referred to as "ERP III" systems (Clegg & Wan, 2013), support functionalities for processes such as EAM (Enterprise Asset Management), PLM (Product Lifecycle Management), PDM (Product Data Management), and QA (Quality Assurance), among other industry-specific functionalities. The additional functionalities can be either embedded in, or integrated with the core ERP system (Panorama Consulting, 2019), and can be supplied either by the same vendor as the core ERP system, or by external organizations via integration. Indeed, the modern ERP systems are far from the closed, complete software solutions they once were, but more like platforms that, in addition to basic business functionalities, enable high connectivity to other systems, and support for external networks with business partner organizations and their software systems.

The modern ERP systems can span between the user organization and its partner organizations and networks by sharing system functionalities in order to facilitate value co-creation (Clegg & Wan, 2013). ERP systems are no longer considered massive, monolithic and closed software entities that are supposed to provide all necessary functionalities for any organization, as the modern ERP systems are highly flexible and connected, with support for a number of integrations to other systems. Nonetheless, the core ERP systems have maintained their critical role in business organizations.

Reflecting the transformation of the functional purpose and business scope of ERP systems, their delivery model has evolved also. ERP systems are increasingly commonly supplied as a service instead of the more traditional, license-fee based delivery model, in which the system was most often installed on-premises.

While delivering enterprise software systems with SaaS model has been commonplace for almost two decades, it has not been the case with ERP systems until the recent decade, mainly because of enduring concerns with information security and system availability (Lenart, 2011). Although delivering an ERP system as a service is becoming more and more common, especially in the small and medium sized enterprises (Johansson & Ruivo, 2013), as of 2019, the on-premises installed solution remained a popular deployment model, as it provides a level of control for the user organizations that SaaS does not. Furthermore, the benefit of lowered operation costs of cloud solution is realized only in smaller user organizations, causing the on-premises deployment to remain a favorable option for large organizations. (Panorama Consulting, 2019.)

Because of the changed purpose and scope of the contemporary ERP systems, some practitioners have debated if the term ERP should be replaced. Furthermore, during the last decade some practitioners have already declared both the ERP systems technologically outdated, and also the underlying concept and value proposition of ERP as legacy, if not obsolete (Forrester, 2019). Indeed, the contemporary ERP systems and their purpose have both become very distant from the systems that were originally labelled as them. This is why some practitioners have proposed new terminology for similar modern system solutions, such as "Digital Operations Platform" (DOP) in order to make distinction between the modern solutions and more traditional ERP systems (Forrester, 2019). Nevertheless, the majority of the enterprise software vendor companies are still currently providing systems that are referred to as ERP to manage their users' core activities. In this understanding, ERP remains a relevant concept and thus replacing the term or updating the terminology otherwise is not necessary.

3.4 Benefits of ERP system

An operational and functional ERP system will provide benefits for business organization in multiple ways and on multiple levels of business operations. Maditinos, Chatzoudes and Tsairidis (2011) reported the following benefits:

- Improved collaboration across functional departments
- Increased business efficiency
- Reduced operating costs
- Facilitation of day-to-day management
- Rapid access to information
- Support of strategic planning

In turn, Shang and Seddon (2000) presented a framework in which the business benefits of an ERP system are classified into five different dimensions, with examples included:

- Operational benefits; such as lower cost of operation and improvements in productivity, quality, and customer service
- Managerial benefits; such as improvements in resource management, decision making, and planning and performance
- Strategic benefits; such as support for growth and product differentiation
- IT infrastructure benefits; such as lower cost of IT services and increased performance of IT infrastructure
- Organizational benefits; such as support for change, empowerment, and creating common visions.

4 RESEARCH METHODOLOGY

In this section, the research methodology of the empirical study is presented, including a detailed description of the research method. Additionally, the data collection is discussed, including the selection of the interviewees, and the process of selecting and conducting semi-structured theme interviews as the main data collection method. Finally, the process of data analysis is discussed in detail.

4.1 Research method

A qualitative research approach enables the researcher to study social and cultural phenomena by combining multiple data collection methods such as observation, interviews, and documents (Myers, 1997). As the phenomenon of interest in this study can be considered both social and cultural, has not been researched extensively, and is relatively recent, selecting a qualitative research method for this study is appropriate. Additionally, as suggested by Cavaye (1996), among others, a qualitative research method was chosen over quantitative, because the main objective of the study is to distill meaning and to gain understanding of a phenomenon instead of primarily concerning measurement and quantification of the phenomenon.

Per Gregor (2006), a theory can be seen as an abstract entity that aims to describe, explain, and provide understanding of the world. For this reason, theories are valuable in generating knowledge for academia. The meaning of theory in the discipline of Information Systems has been perceived in different ways, i.e. theory can be viewed as "a statement how something should be done in practice", as "a statement providing a lens for viewing or explaining the world", and as "a statement of relationships among constructs that can be tested". A theory that is considered analytic is the most basic type of theory and analyzes "what is" as opposed to explaining causality or attempting prediction. Analytic theories can manifest in the form of frameworks, taxonomies, and classification schemas. (Gregor, 2006.) As the objective of this study is to describe and analyze a phenomenon, the resulting body of knowledge can be described as an analytical theory. Analytical theories have been also referred to as descriptive theories, but that term is not completely appropriate, as analytical theories go "beyond basic description in analyzing or summarizing salient attributes of phenomena and relationships among phenomena" (Gregor, 2006; p. 623). The relationships specified are not explicitly causal, but classificatory, compositional, or associative instead (Gregor, 2006).

However, in order to gain additional understanding of the phenomenon, some components of explanatory theory, namely causal explanations between theory constructs, were also included, which will produce a novel theory in form of a conceptual framework of the Continuous Delivery adoption challengees.

4.2 Data collection

Semi-structured theme interviews were selected as the primary method of data collection, as they facilitate interaction between the particitipants, potentially leading to more in-depth information collection when compared to other interview methods. When used correctly, the qualitative interview is a very powerful technique for data collection (Myers & Newman, 2007). The interview data was obtained from persons representing companies that fulfill certain criteria, namely Finnish ERP system vendors, with those persons preferably holding a position of product development manager or similar. In addition to the semi-structured theme interviews, archival data, both public and private in form of marketing materials, public financial data, and internal training documents of the organizations, were used as a secondary data source.

4.2.1 Selection of interviewees

While selecting potential interviewees for the study, the organizations they represented were considered first in order to obtain suitable interviewees with necessary experience and knowledge of the phenomenon of interest. The following criteria were set for the organizations to be contacted in order to obtain interviewees:

1. the company practiced, or was in the process of adopting continuous software development and delivery methods
2. the company operated in the ERP system market as a vendor
3. the company was classified as small or medium sized enterprise (<250 personnel; ≤50M€ turnover; ≤43M€ balance sheet total)
4. the ERP system the company supplied was developed exclusively by them

5. the company operated predominantly in the Finnish ERP market

The first two requirements were set in order to obtain applicable and relevant information to address the research problem, while the third one was set in order to exclude organizations that provide system products developed by other organizations, thereby functioning more as an intermediate agent providing additional support rather than as the initial developing organization of the system that faces the challenges and problems related with the adoption of Continuous Delivery practice. The last requirement was set in order to control extraneous variation, to enhance external validity, and to help in defining the limitations for generalization of the findings, as suggested by Eisenhardt (1989).

Following the requirements above, the organizations that potentially could provide interviews were selected among Finnish ERP software vendor companies. As the phenomenon of interest is the challenges related with Continuous Delivery practice adoption, only companies with their own product and product development were considered, excluding vendors that supply and maintain ERP system products developed by other organizations. As obtaining interviews from companies can be difficult (Myers & Newman, 2007), the combination of limited range of potential companies and the general difficulty in obtaining interviews resulted in limited amount of potential companies to provide the interviewees. A total of 15 companies were contacted by email, of which nine did not respond. Two companies declined briefly, and finally, four companies approved their employees or an employee to be interviewed. This was not completely unexpected, as it can be difficult to obtain interviewees, especially if there is no prior connection with the contacted organization. Moreover, there are certain other factors that can affect the outcome of the interview request, such as inappropriate level of entry (Myers & Newman, 2007).

Persons holding the title of head of product development, product development manager, or other comparable role were initially targeted, as persons holding those positions presumably have expert knowledge of all the development activities in their organization, including the status of Continuous Integration and Continuous Delivery practices, while also having a clear understanding of their core business operations and practices, including their product portfolio, market position, and strategic outlook. Eventually five persons representing four different organizations were selected to be interviewed, with each interview to be conducted individually with no information shared between the interviewees in order to maintain rigor and trustworthiness of the research, and the confidentiality of the interviewees. In order to further increase trust and level of confidentiality, the interviewees and the companies they represented were decided to be left anonymous and each company and interviewee given a simple alias in form of a code instead. The interviewees and their roles in their organizations are summarized in the Table 5.

TABLE 5 Interviewees and their roles

Interviewee ID	Company ID	Role	Interview duration (minutes)
A1	A	head of product development	76
B1	B	head of product development	41
C1	C	head of product development	48
D1	D	technical lead	34
D2	D	test engineer	36

4.2.2 Semi-structured theme interviews

Semi-structured theme interviews were selected as the primary data collection method, as they can provide deeper knowledge about the phenomenon of interest as opposed to more rigid interview methods, including structured interviews with a set pattern of questions. A highly structured interview method can guide the interview into a certain direction as opposed to a more natural setting, which can lead to a situation where data relevant to the research problem remains uncovered during the interviews. Furthermore, as a deductive approach for theory building was selected for this study, the semi-structured interviews allow the units of analysis, i.e. codes, patterns, and themes, to be compared and mirrored with existing theoretical knowledge, which is favoured for the deductive approach to be successful. Finally, as opposed to unstructured, or "free-form" interviews, semi-structured theme interviews facilitate retaining research focus to a defined area of interest. The initial interview themes representing the possible Continuous Delivery adoption challenge areas were drawn from the existing software engineering literature, and were the following:

- Release management (e.g. Barqawi et al., 2016)
- Strategic release planning (e.g. Svahnberg et al. 2010)
- Testing, deployment & support (e.g. Claps, Svensson & Aurum, 2015)
- Release scheduling (e.g. Greer & Ruhe, 2004)
- Release content (e.g. Fricker & Schumacher, 2012)

After the first interview was conducted, it appeared that the preselected themes to guide the data collection were not ideal for addressing the research problem and therefore had to be revised in order to enable more focused interviews. After the initial analysis of interview data, it was also noted that certain themes were overlapping with each other or were otherwise unnecessary for addressing the research problem or answering the research question. After consideration, the new interview themes [to direct the data collection] were adopted from Continuous Integration and Continuous Delivery problem themes presented by Laukkanen et al. (2017a), which are the following:

- Build design
- System design
- Integration
- Testing
- Release
- Human and organizational
- Resources

The theme *Release* simultaneously included and replaced the previous themes *Release management*, *Release scheduling*, and *Release content*. As *Strategic release planning* can be recognized more as an organizational than purely developmental activity, it was included into the *Human and organizational* theme. For the rest of the interviews, the themes presented above were then discussed with the interviewees. Although it was the initial object of the interviews, themes were not necessarily discussed in any particular order. This was because at times, the natural flow of the conversation led to themes being discussed in different order. As the semi-structured method was chosen for the interviews, no defined questions were formulated, but a rough outline for items related to each theme instead, with some discussion and topic prompts also included in order to facilitate conversation, if necessary. The outline of the interview guide is presented in the Appendix 1.

The interview data was collected over an eight-month period between June 2019 and March 2020. Each of the interviews were conducted remotely with telecommunications software such as Microsoft Teams and Skype and the interviews were recorded in order to allow further data analysis. Remote connections were preferred for additional convenience in scheduling and to avoid unnecessary and time-consuming travelling, as the case companies and interviewees were located in different parts of Finland. The interviews were conducted individually with no information shared between the interviewees in order to maintain rigor and trustworthiness of the study, and also to maintain confidentiality of the interviewees and their organizations.

For practical reasons, the interviews were conducted in Finnish. Because both the interviewer and each of the interviewees were native Finnish speakers, it was deemed more natural to use Finnish rather than English. The interviews were planned and scheduled to have duration of approximately one hour. The longest of the interviews exceeded this, with the final duration being 1 hour and 16 minutes. In contrast, two shortest interviews lasted 34 and 36 minutes, resulting in a median duration of 41 minutes. The average duration of the interviews ended up being 47 minutes, which was considerably less than planned, but after review, still sufficient time to gather a satisfactory amount of data.

4.3 Data analysis

There are numerous methods for analyzing text data, such as ethnography, grounded theory, phenomenology, historical research, and qualitative content analysis. Qualitative content analysis was selected as the method of data analysis, as it provides knowledge and understanding of the phenomenon of interest. (Hsieh & Shannon, 2005.) According to Hsieh and Shannon (2005), the qualitative content analysis can be classified into conventional, directed, or summative approach depending on how the initial codes in the analysis are developed. For this study, a directed content analysis approach was selected. In the directed content analysis either an existing theory or relevant research findings act as guidance on how the initial codes are defined (Hsieh & Shannon, 2005). The process of complete data analysis is presented in Figure 4.

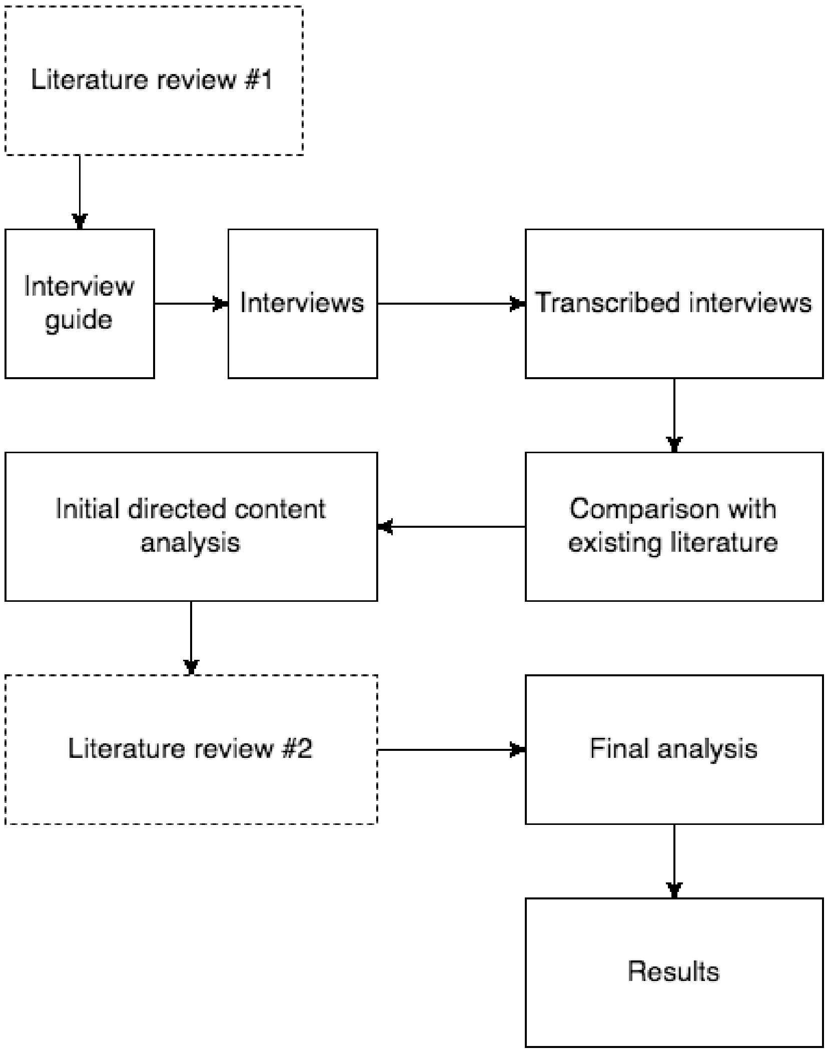

FIGURE 4 Qualitative directed content analysis process of the study

After the interviews were conducted, the recordings of them were transcribed in their entirety, as in addition to facilitating the data analysis it also increases the credibility and auditability of a study (Sarker, Xiao & Beaulieu, 2013). In addition to the interview themes guiding the data collection, they did also guide the data analysis, as the initial data analysis was conducted by deductive approach, in which research data is analyzed against existing theoretical knowledge, with the objective of finding meaningful patterns and codes reflecting and matching the themes found in the existing literature. When applying a directed content analysis approach, an existing theory or prior research can be used to develop the initial coding scheme before initiating data analysis (Kyngäs & Vanhanen 1999, as cited in Hsieh & Shannon, 2005). In the directed content analysis, the coding of results can start immediately with predetermined codes drawn from existing previous research. If there is data that cannot be coded initially, it is identified and analyzed later to determine if they represent a subcategory of an existing code or a new, separate category (Hsieh & Shannon, 2005).

After the initial data analysis, an additional literature review of related existing research was conducted, after which the transcribed interview data was analyzed again. As data analysis progresses, additional codes can be developed, and the initial coding schema can be reviewed and refined accordingly (Hsieh & Shannon, 2005). After the second round of data analysis, it became apparent that additional codes, categories, and themes emerged, and as a result the themes were revised and refined again. After this cycle of iterative textual analysis, the final five themes, in which the results in form of Continuous Delivery adoption challenges are categorized, were identified.

5 RESULTS OF THE EMPIRICAL STUDY

In this chapter, the results of the empirical study are presented, including a description of the interviewees and the organizations they represented. As the use of quotations conveys level of "richness" that is not usually achievable with quantitative methods, and thus appear to be valued by the audience of qualitative research (Sarker, Xiao & Beaulieu, 2013), quotations by the interviewees are included in the presentation of the results. Because the quotations were translated from Finnish transcriptions of the interview data, the original Finnish quotations are presented in the Appendix 2. A total of 12 Continuous Delivery adoption challenges for ERP system vendors were identified and classified into five themes: system design and architecture, integration and deployment pipeline, testing, release, and organizational challenges.

5.1 Organizations and interviewees

The interview data was collected from five persons representing four Finnish small to medium sized ERP vendors. The key characteristics of the companies are summarized in Table 6. As frequency of releases is one possible way to measure success and state of the Continuous Delivery adoption, the reported frequency of ERP system releases is included in Table 6.

5.1.1 Organizations

Each of the four organizations were Finnish ERP vendors with their annual revenue ranging from 2 to 10 million euros and number of personnel ranging from 20 to 100, classifying them as small and medium sized enterprises (Eurostat, 2020). Every participating company in this study was in a different situation and stage with their Continuous Delivery adoption. As expected, the smallest organization, Company C had taken only the initial steps towards Continuous Delivery, as their release activities were still conducted mostly manually. Company A had developed their Continuous Delivery capabilities furthest, but had still selected to hinder their release process intentionally. Companies B and D had already applied deployment pipelines in their product development, but they were only partly functional, as testing activities were still conducted mostly manually, albeit for different reasons. B1 explained their situation:

> ”There are processes for how we do testing, but we haven’t had much time to create reasonable automated tests. We have them for some specific purposes, but not to any greater extent.” (B1; 1)

All of the companies used agile software development methods, which is an important precedent for adopting continuous practices (Olsson et al., 2012). Each of the companies reportedly used some variation of the Scrum framework, but none of the interviewees reported to use it in its standard form, but customized to the most suitable form, that was discovered through practice instead. Companies A, B, and C used scheduled, fixed-length sprints with duration of approximately two to three weeks, or 10 to 15 business days, while Company D had transferred from scheduled sprints into feature-defined sprints, in which completion of a releasable feature determines the duration of a sprint. D1 elaborated on their solution:

> ”At present we don’t plan what we are going to do in a period of time, but instead we plan a feature and how we are going to do it, how long it is going to take, and when we are going to begin, so we don’t have sprints of a week or two. In a way, sprints happen within the development of a feature. --- They can last week or two or whatever, but, in a sense, we plan them from the viewpoint of the planned outcome.” (D1; 2)

In that sense, it was not easy to define the status of Continuous Delivery practice adoption in the companies, other than by gaining information from the interviewees. This succeeded in differentiating ways, as for example, the meaning of Continuous Delivery had to be initially explained to one interviewee. Eventually, they appeared to be already familiar with the concept, but not with the established terminology. This reflects the issues with non-standard terminology and vocabulary discussed previously in the chapter 2.6. Each of the companies clearly understood the importance of Continuous Delivery for their development and release activities, and overall organizational performance and competence. However, C1, representing the smallest company interviewed, emphasized the aspect of the company size regarding the usefulness of the practice. C1 said:

> ”If we have a team of 10 people here, they can move them even manually. You can ask everyone individually, at what point a certain ticket is, and they can move it manually. But if you consider that there were 100 developers, for example, it would be a lot more challenging to keep that system up to date.” (C1; 3)

All of the companies reportedly offered their ERP system as an on-demand solution, or service, with companies A and B offering their product also in other ways, including more traditional on-premises or cloud-hosted installation, or a combination of the two. Both companies reported to also provide their systems as license-based installations instead of a service, with on-premises installations remaining a prevailing option alongside cloud hosted installation, even though the cloud hosted solution had become an increasingly popular option for clients of both companies. Unlike other companies, Company B did not actually externally market their system as an ERP, but as a core part of their 'business software platform' consisting of seven different software products, one of which covered the traditional ERP functionalities and was internally regarded as their core software system product.

TABLE 6 Companies and their key figures

Company ID	A	B	C	D
Revenue (2019, M€)	10	10	2	5
Number of employees	+70	+100	+20	+50
Frequency of releases (ERP)	Monthly	Trimonthly	Monthly	Biweekly (target)

5.1.2 Interviewees

Interviewees A1, B1, and C1 held the position of head of product development in their companies, with A1 and B1 of them having similar roles and areas of responsibility. C1 had a differing role because of the smaller size of the company they represented, with their area of responsibility also covering the project management activities of the development team, which was responsible for development of their system product. As the head of product development of Company D was unavailable for an interview, two persons with the roles of technical lead (D1) and test engineer (D2) were interviewed instead. Instead, a person holding the position of technical lead was deemed to have sufficient information and knowledge regarding the status of the Continuous Delivery practices of the organization for the scope of this study. A person holding the position of test engineer of the same company, D2, also provided more in-depth information about the testing activities and challenges related to them.

5.2 System design and architecture challenges

As Continuous Delivery relies heavily on automated build, testing, and deployment, adopting the practice is considered challenging. This is especially the case with an existing software product as it often also requires changes both in the system design, and the development organization, which can be not only difficult, but also very expensive (Laukkanen et al., 2017a.) This was a rather familiar situation for three of the four companies (A, B, C), as their core ERP product, or at least some components and functionalities of it, preceded the Continuous Delivery paradigm, or at least the adoption of it in those organizations. Additionally, Company D had two core system products of which only another was developed recently and thus with requirements of Continuous Delivery in mind. The following two system design and architecture challenges were identified: unsuitable architecture, and system scale and complexity.

5.2.1 Unsuitable architecture

Architecture that is considered unsuitable for Continuous Delivery is often described as monolithic, coupled, unnecessarily capsulated, or consisting of multiple branches of code. (Laukkanen et al., 2017a) These characteristics are also generally associated with the terms 'legacy software' and 'legacy architecture'. By one definition, legacy software means "programs which have been developed with an outdated technology". (Sneed, 2006; p. 1) By the same definition, most of the current software is considered outdated, because the prevailing innovation cycle of software technology is less than five years (Sneed, 2006), while the life cycle of software is generally longer, especially in the case of critical enterprise systems such as ERP. Furthermore, legacy systems are considered mission critical, expensive to maintain, inflexible to changes, run on obsolete hardware, and are difficult to enhance and integrate with other systems. (Gholami, Daneshgar, Beydoun, & Rabhi, 2017) In addition, even modern systems based on web-enabled architecture, which have been developed with latest technologies available, can be considered legacy systems if they do not satisfy new and emergent business requirements. However, systems considered 'legacy' could still provide positive return for an organization, as they support business processes and maintain organizational knowledge (Gholami et al., 2017.)

Most of the companies had developed their current ERP system software product, or at least some parts of it, on a foundation of an already existing software, that could be described as 'legacy', namely software that is installed on-premises and accessed primarily by desktop PCs. For example, B1 regarded their ERP system, even though considered 'legacy', as their core system product on which their other software functionalities, products and services were built upon. B1 elaborated on their situation:

> "Our oldest system, maybe a bit of a legacy system already, functions as our ERP system, and in a way, is the focal point of our all activity. --- And then, over the years, we have introduced all kinds of add-on products around this ERP system." (B1; 4)

Maintaining a large and complex enterprise system such as ERP to conform fully to modern specifications and requirements is not only difficult, but can also most often prove to be impossible, as the whole system cannot be updated concurrently, leading to older system components becoming legacy software. C1 discussed how a large system cannot be ever fully up to date, as the older system components and functionalities become technologically outdated at the same rate new ones are planned and developed:

> "If you finish some larger new part, the system starts to become outdated from the other end, technologically. --- But it's just a fact that has to be accepted when the system is the same that has been developed for over 10 years, and is still developed, that it's [legacy software] in multiple ways, the use of obsolete methods can be seen there." (C1; 5)

5.2.2 System scale and complexity

The more massive and complex a software system is, the more difficult it is to implement, upgrade, and maintain. This is especially true in the case of a system that is delivered continuously, as frequent changes in form of fixes, upgrades, and new functionalities have the potential to cause unintentional system behaviour in form of glitches and other failures, unless they are reviewed and tested extensively. Furthermore, the changes can affect the business aspects of the user organizations, which can cause further organizational issues experienced by the users. A1 talked about the general challenges that are caused by the scale and complexity of an continuously evolving ERP system:

> "We are producing an ERP system, which are really multifaceted and extensive systems, and there are changes made constantly --- So those combinations which have to be tested, different languages, different platforms – especially now as we have this online version – there are gigantic amounts of ways and routes of use." (A1; 6)

Due to the large scale and increased complexity of ERP systems delivering them continuously is different and considerably more challenging than of less critical software applications and systems, e.g. consumer software products. According to B1, the users of consumer software, such as relatively simple mobile applications, are usually only delighted and eager to receive new features frequently, but updating an ERP system is a substantially more difficult process. B1 discussed the additional effort that is required by both them and their customers when updating their core system:

> "The update process [of ERP system] differs 'a little' from, let's say, that of some consumer product. --- In our case the update often requires training and a visit from our project manager, so they can explain how the upgrade affects the business activities

of the client. --- Those usually come with an additional price, because they usually affect the business operations of the client organization in multiple ways." (B1; 7)

5.3 Integration and deployment pipeline challenges

As practicing Continuous Integration and utilizing a deployment pipeline are critical preceding steps for successful Continuous Delivery adoption (Olsson et al., 2012), it is important to to achieve sufficient capabilities with both of them in order to succeed with Continuous Delivery adoption. Companies A, B, and D practiced Continuous Integration in their development process and utilized an operational deployment pipeline, or multiple of them for different system functionalities. While company B reported to utilize multiple deployment pipelines, none of them were fully automated, as there were still some aspects, such as testing automation, under development. In total, two integration and deployment pipeline challenges were identified: revision control and code freezing practice.

5.3.1 Revision control

In the context of this study, revision control refers to the activities related to the management of changes in the software source code, including Continuous Integration practices. As ERP systems are large scale, high complexity software, their revision control requires remarkable efforts.

A1 discussed how in the past, prior to implementing a CI server, code changes and new features could be unnoticed for everyone except the developer responsible for it and the customer who requested it, leading to a situation, where unidentified and untraceable code changes could be noticed only years later. However, more recently with a CI server in operation, changes could be traced successfully even if there were thousands of them. D2 noted that a large workload for QA personnel occurred when the development and master branch were attempted to be merged. Companies that applied Continuous Integration reported to apply a specific branching process, for example, Company A had chosen the 'no branches' discipline as their branching strategy, as A1 mentioned:

"We actually retired our master branch, now we have only this integration branch, into which we merge commit our code changes." (A1; 8)

Moreover, Company B reportedly applied a specified branching method, but had to deviate from it occasionally, especially in situations where the customer was in the process of implementing their ERP system. B1 explained the situation:

"We also have those kinds of cases, it's the nature of our product, as it is an ERP system and the customer implementations are really long, they can last a year. So as the

> product [is] implemented, it's not reasonable to go with a certain version tag with that customer." (B1; 9)

Instead, as B1 reported, they wanted to deploy their newest, or daily development version for those customers in the process of implementing the ERP system by having a dedicated development branch. Still, the principal option for them was to use the specified branching method, which was the procedure of using version labels. However, deviating from the specified branching method can potentially cause issues with revision control.

5.3.2 Code stabilization

In some situations, the software code requires an additional stage of stabilization before release. One of the code stabilization methods is known as freezing. The practice of code freezing means that the code mainline is "freezed", as in that no additional code changes are permitted before the release (Laukkanen, Paasivaara, Itkonen & Lassenius, 2017). Code freeze is a useful method to mitigate challenges and risks related to frequent releases, but as it contradicts the basic principles of Continuous Delivery, namely the principle of always releasable code, it can be seen as a challenge, or even a barrier, for adopting and practicing Continuous Delivery. Therefore, reducing the duration of code freeze periods, or eliminating them entirely, will effectively facilitate the adoption and practice of Continuous Delivery.

Companies C and D applied the practice of code freezing in their development cycle in order to increase stability of the software builds before releasing. C1 talked about applying the freeze in order to achieve a fixed release schedule that is known to their customers beforehand. D2 also explained their practice of code freezing, to which they referred to as version freezing:

> "One week before it [release] we set a content freeze, so new code is not allowed, we revise what is already in there in case something is found during testing, so we can maintain the release schedule to be always on a certain date." (C1; 10)

> "We [QA personnel] freeze the version at some point before it's set to be released. So during the freeze programmers have a 'block' regarding whatever they do, their pull requests are not merged, at least not into the master branch." (D2; 11)

However, even companies with fully operational deployment pipelines, including Company A, were still stalling the delivery process on purpose, mainly for precautionary reasons and increased release stability, A1 discussed and explained their decision:

> "In a way, we have all that automation, and everything is ready for us to release without anyone having to go manually through any stages so the packages would be released to users of our platforms. But we have decided to set up certain stops that include small checkups, to give us time to catch breath and make sure that everything is alright, before we proceed to release." (A1; 12)

5.4 Testing challenges

Testing challenges refer to the challenges that are related to the testing of the software and other activities conducted mainly by the Quality Assurance personnel. Each interviewee discussed testing activities and the challenges related to them in great detail. The testing challenges reported by the interviewees were mostly related to test automation and the excessive amount of time required for testing, but not exclusively, as there were also challenges caused by the considerable variations required to be addressed in the testing activities. The following two testing challenges were identified: time-consuming testing and disparity of testing.

5.4.1 Time-consuming testing

In the context of this study, time-consuming testing refers to the excessive amount of time that is required to conclude all the necessary test activities for software to be in a releasable state, or in other words, the duration of necessary testing activities can be considered unsustainably long. This is caused by a variety of reasons, such as unsuitable system architecture, lack of resources, and insufficient testing strategy. Moreover, time-consuming testing is highly associated with the lack of test automation. However, pursuing complete test automation is not sufficient by itself to solve the problem, as testing could still take impractical amounts of resources such as time or testing staff. D2 confirmed the fact and discussed how the challenge of obtaining sufficient test coverage affects their testing activities, as everything cannot be tested:

> "We can't test everything our software does, as even if we had made automated tests for everything, executing them would take a month." (D2; 13)

> "The [testing] coverage is achieved by finding so-called 'blockers'. So if – if we identify features that have to be functional, e.g. storing data or adding a new customer – if those features are not functional, that version is not releasable." (D2; 14)

Moreover, not all testing activity can be automated, especially when developing a mission critical enterprise system such as ERP instead of relatively less complex commercial software. D1 described the situation with the requirements of testing for an ERP system:

> "Even if we got our end-to-end tests or acceptance tests to any form, we probably can't or won't ever dare to deliver without manual testing, so well, some kind of manual testing will remain in any case. It would be different if we developed some simple phone app or something else, the testing could be done more or less exclusively with automated tests." (D1; 15)

5.4.2 Disparity of testing

Disparity of testing refers to the wide array of different types of test cases and testing activities that are required during the development of an ERP system, which are caused partially by their large scale and high system complexity. A1 discussed the major challenge caused by the disparity. Furthermore, as noted by B1, different types of software functionalities and components of the system require different types of testing. They said:

> "In a sense, those variations in the testing that we are supposed to take into account, that seems to be an almost overwhelmingly big issue, in a way." (A1; 16)

> "The challenge is that the systems in our business software platform are different not only technologically, but also in their nature; so there is more legacy-like Windows desktop user interface, which is considerably different to test in an automated way than, let's say, some interface functionality using APIs. So from a testing viewpoint, those are complete opposites. That is the challenge, there are so many, and so different types of use cases." (B1; 17)

B1 also discussed how the large amount of different parametrization and configuration options in the system affects the testing activities, as according to them there are as many ways to use the system as there are its users:

> "This is what causes atrocious challenges for testing, especially if we are talking about automated testing, because it would take automated testing staff of like 200 people, who would maintain those tests and keep them synchronized with projects to make it work, so in a way that makes absolutely no sense." (B1; 18)

Not unexpectedly, they also confirmed the fact that not all testing cannot be automated, and discussed how automated testing can still be applied to some aspects of the system development: B1 said:

> "My policy has been that certain basic functionalities, that remain roughly unchanged from version to version, should be automatically testable. --- Each one of our parameterization options – or what other use cases comes to my mind – we simply can't test, or above all, automate their testing. So this is the challenge." (B1; 19)

It should be noted that the system changes constantly, especially in the case of Continuous Delivery practice taking place. As the system evolves further, constantly differing test cases, activities, and strategies are required, and why test automation is difficult to implement and maintain. D2 explained:

> "When the system develops further, the test cases have to change too. So each test case – this is the problem, why automated tests are really heavy, and difficult to create too. They will surprisingly quickly cause a sort of 'maintenance hell'. And if they are not maintained, they will not test the right things, after which it doesn't matter if the test is valid or not." (D2; 20)

5.5 Release challenges

While in the practice of Continuous Delivery frequent releases are emphasized, it is not straightforward how, and when they should be carried out, and what their contents and scope should be. There are multiple factors that affect those decisions, including varying requirements of the users, available development resources, strategic factors, and legislation. The following three release challenges were identified: customer requirements and preferences, domain restrictions, and release planning.

5.5.1 Customer requirements and preferences

Customer requirements and preferences refer to the requirements set by the user organizations that system vendors have to address when practicing Continuous Delivery. Customer requirements and preferences appeared to affect the release activities of the case companies in multiple ways. Each of the interviewees emphasized that the development activities, including releases, are planned and performed with the customers' requirements and preferences in mind. In other words, the impact of the customers is considered vital in the overall development process. A1 summarized what they thought to be one of their largest challenges regarding the customer requirements and preferences:

> "Maybe it is exactly how much the customer demands have increased and how the technology keeps advancing constantly." (A1; 21)

The case companies had actually encouraged the customers to actively take part in the development process. This was exhibited in a few different ways. For example, Company A had created a portal in their system for users to suggest and vote for upcoming features, while Company D had organized specialized workshops for customers in order to receive more accurate, and even notably, almost immediate feedback on the features and functionalities under development.

5.5.2 Domain restrictions

Domain restrictions refer to limitations caused by certain characteristics of the user organizations' area of business, i.e. regulations, legislation, and other industry requirements. Some application domains are less suitable for Continuous Delivery than others. For example, safety-critical systems, including a number of ERP systems, do not facilitate Continuous Delivery (Laukkanen et al., 2017a), yet upgrades can be planned and delivered in a quick and reliable manner, even in strictly regulated and controlled environments (Ebert et al., 2016) by adopting continuous practices. In fact, Continuous Deployment and Delivery might be prohibited by a service-level agreement (SLA) or other contract, which is notably common in strictly regulated industries, such as defense and healthcare. As Company D developed their ERP system for the healthcare industry, D1 confirmed that practicing Continuous Delivery might not be even possible for them, even if they had the necessary capabilities for it. They discussed the fact that they cannot always release without additional acceptance testing conducted by the user organization, effectively obstructing the practice of releasing frequently. D1 said:

> "Often our contracts disallow Continuous Delivery with certain clients, and releases have to go through their own acceptance testing first." (D1; 22)

5.5.3 Release planning and prioritization

Size, scope, and purpose of a release can vary, ranging from small software patches to completely new features and functionalities, or even larger system revisions and updates that can have effects even at the architecture level. For that reason the releases have to be managed with activity referred to as release planning. Prioritization, in turn, is an aspect of release planning, which is concerned with the decisions on what and when to release, and especially in which order. While larger system revisions require more careful planning than small patches, it is important to plan all releases in some way, regardless of the type of release. Each of the companies reportedly planned their releases beforehand, also on the strategic level, with most interviewees referring to that activity as 'road-mapping', which is an alternative term for strategic release planning (Svahnberg et al., 2010). A1 said:

> "From the viewpoint of development it's rather simple, we release packages all the time, but in the case of some larger or more important items, we constantly consider when we should release them, we actually have a road-map, in which larger developmental activities are scheduled and prioritized." (A1; 23)

As the intention of software releases is to answer a stream of requirements that are presented to the development organization (Fricker & Schumacher, 2012), both externally by the users, and internally by the product development team, and product department in general, these requirements need to be carefully evaluated and selected in order to plan the development of future releases. As not each requirement can be addressed at once, an issue of prioritization arises.

5.6 Organizational challenges

While Continuous Delivery can be seen essentially as a technology-focused method, organizational factors can also affect its adoption, with certain organizational characteristics and practices potentially preventing its successful adoption. On the contrary, other organizational practices can greatly facilitate the adoption of Continuous Delivery practice. According to the interviewees, organizational challenges were considered to be more straightforward to solve than other challenges, but they were not regarded as trivial by any means, as the interviewees discussed serious efforts that were required to achieve satisfactory results within this aspect of Continuous Delivery adoption. Although the interviewees reported organizational and resource-related challenges to be already mainly resolved in their companies, the following three organizational challenges were identified: coordination and collaboration, communication, and resources.

5.6.1 Coordination and collaboration

Continuous Delivery requires different kind of approach, and ways of working from the development team, including the developers, QA personnel, and project management. Restructuring development processes and teams is not straightforward, as there are multiple factors affecting the restructuring decisions. A1 elaborated on the impact that restructuring the development and release processes had initially on the development team members, and the relationship between them and the management:

> "In the beginning, the developers felt somewhat negative. They felt that they were being stalked, and doubted if they were relied upon, even though they felt that they worked hard." (A1; 24)

According to A1, however, the situation improved and soon everyone realized the improvements the restructuring had caused, as having a clearly defined development process with distinctive stages had received acknowledgement from everyone in the organization. Still, they discussed the remaining challenges caused by the organizational structure, as they had separate departments in their organization that operate from their own viewpoint, and sometimes it is difficult to get everyone from different departments involved.

Indeed, the organizational structure can affect the coordination and collaboration within the development teams and between other departments and stakeholders in a negative manner. B1 discussed the importance of having a functional organizational structure and coordinated development schedule in order to facilitate collaboration within the organization and development teams. They said:

> "From my viewpoint, the most crucial challenges related to personnel and organizing have already been somewhat resolved in our company, as we have finally managed to get all of our teams to work within the same sprint cycle and everyone has in a way adopted the sprint paradigm into their daily activities." (B1; 25)

5.6.2 Communication

Communication is an important factor attributing to performance of development teams, and overall success of the organization. D2 discussed the challenges of communication between different people involved in the development process and the issue of everyone having their own, distinct viewpoint to the overall software development process. He emphasized the importance of the ability to perceive and consider issues from different perspectives, especially from the viewpoint of other stakeholders involved. D2 talked from the viewpoint of QA personnel and how they have to approach their testing activities from different perspectives:

> "Our job, in a way, is to remain sceptical about what the client wants and if they understand what it [the system] should do. And that mirrors back to us; first we have to test it in a way a developer would, after which we test it in a way we want to. And after that, we try to break, in the same way how the client would use it." (D2; 26)

B1 mentioned how varying amounts of work experience between the development team members affects the communication between them. It had taken serious efforts in order to achieve a sufficient level of communication between the team members. Indeed, while the lack of "common language" is an almost universal problem in the business world, the software industry can be affected proportionally even more as new programming languages, frameworks, and libraries, and development tools, practices, and methodologies as well, are introduced relatively frequently. B1 summarized the challenge:

> "It has been a surprisingly major challenge to get this kind of a large development staff to speak the same language." (B1; 27)

5.6.3 Resources

The interviewees discussed the considerable efforts they had been putting towards the development and adoption of the Continuous Delivery practice. In addition to financial resources and personnel, correct software development tools are important resources not only intrinsically, but it also requires remarkable resources from the organization to obtain or internally develop sufficient development tools to facilitate the adoption of Continuous Delivery (Shahin et al., 2017). D1 also emphasized the significance of correct development tooling for the overall development and release processes, and how the practices act as prerequisites to other subsequent practices:

> "In a sense, it's the tools that make it Continuous Integration and Delivery, or it is the [Continuous] Integration that makes, or at least is supposed to make, Continuous Delivery safe. It doesn't always go that way, but essentially Continuous Integration is required to be able to release frequently, or at all." (D1; 28)

6 ANALYSIS OF THE RESULTS

In this chapter, the results of the empirical study are analyzed and discussed. Additionally, the results are compared to previous research in order to discuss similarities, differences, and conflicts between them. Finally, the challenges of Continuous Delivery adoption are summarized in the form of a conceptual framework.

The identified challenges that were classified into five themes are not disconnected from each other. Additionally, the themes are interconnected in many ways, and could be classified differently depending on the interpretation of the research data. As the chosen method of analysis was directed content analysis, the final result themes are similar, but not identical as those presented by Laukkanen et al. (2017a). The differences in the themes can be explained by at least two reasons. Firstly, the Continuous Delivery problems and challenges identified in the previous literature were related to general software, but the phenomenon of interest in this study was strictly a certain type of software, ERP systems. Characteristic differences between ERP systems and other software, such as its critical role for users, can expectedly lead to differing Continuous Delivery adoption challenges. Secondly, as the directed content analysis can potentially refine, extend, and enrich a theory or prior knowledge (Hsieh & Shannon, 2005), the themes identified in this study can be viewed as a refined classification of the Continuous Delivery adoption challenges.

6.1 Continous Delivery capabilities of the organizations

Perhaps the most straightforward way to assess the status of Continuous Deployment adoption would be using a binary scale of "yes" or "no" to denote whether a fully automatic deployment chain exists, or if there is one or more manual steps. (Leppänen et al., 2015) From this viewpoint, Continuous Delivery adoption could be measured in the same way, by answering "yes" or "no" to whether the organization has the capability to maintain the software in a constantly releasable state, or not. Such an absolute scale, however, wouldn't accurately illustrate situations where organizations have taken steps toward automated deployment or Continuous Delivery without yet utilizing a fully functional deployment pipeline (Leppänen et al., 2015) or other sections of the complete infrastructure required for adoption of Continuous Delivery practice.

In order to evaluate the status of Continuous Delivery adoption in a more accurate way compared to the binary scale, Leppänen et al. (2015) used multiple factors to assess the Continuous Delivery capabilities of companies they interviewed in their case study, including "fastest time for a code change to propagate to production", "cycle time to potentially deployable software", and whether there is an "automatic chain to potentially deployable software", or not. In the scope of this study, access to such information was limited, as the interviewees either did not disclose the details, or preferred not to give rough estimates. Each interviewee did, however, discuss the frequency of their customer releases, which ranged from two weeks to four months. While release frequency can imply the status of Continuous Delivery adoption, it is not sufficient measurement on its own for assessing Continuous Delivery capabilities, as the organizations may simply choose to release less frequently for varying reasons, despite their actual capabilities. The frequency of releases is not the deciding factor, but the ability to deploy at will (Neely & Stolt, 2013).

6.2 System design and architecture

The following two Continuous Delivery challenges related to system design and architecture were identified in the study: unsuitable architecture, and system scale and complexity. The identified challenges are also highly interconnected, as the former can be seen as a factor accounting for the latter, and vice versa.

Consistent with the findings of extensive literature review conducted by Laukkanen et al. (2017a), Continuous Delivery adoption challenges related to the system design and architecture emerged as one of the most prevalent challenges reported in the interviews. This was not unexpected, as the companies had begun to apply Continuous Delivery practices to their development activities only relatively recently, while at the same time retained the same core software product, or parts of it, that had been developed prior the introduction of Continuous Delivery practices. The only exception to this was Company D, as they had a more recently developed product in addition to a system solution that could be regarded as legacy software.

The challenges related to system design and architecture are relatively difficult to solve, as they may require large-scale revisions to the entire software system, including the core architecture level. Reconstructing the system architecture of an existing software system is considerably more difficult compared to designing, defining and revising architecture for a system that is in the initial stages of its life cycle, such as planning or development stages. One way to mitigate challenges caused by both unsuitable architecture and system scale and complexity is to divide the system into smaller modules or other functional components, as the modularized system units can be tested and deployed independently. (Laukkanen et al., 2017a) However, unnecessary coupled component architecture can actually lead to increasingly unsuitable system architecture, as it limits the practice of Continuous Integration (Bellomo, Ernst, Nord & Kazman, 2014), and thus also the practice of Continuous Delivery, as the former is required for the latter (Olsson et al., 2012). This means that there is no universal standard regarding the most advantageous level of modularization and encapsulation of the system architecture, but it depends on a large number of factors, including functionality requirements of the system and resources of the developer organization.

It is important to understand that, as noted by Laukkanen et al. (2017a), unsuitable architecture is not a problem on its own, but it causes different types of problems and challenges with testing, development, and deployment. Therefore, addressing challenges in system design and architecture does not only enhance system performance, but also facilitates the resolution of the challenges related to the testing, development, and deployment activities. Similar to the unsuitable architecture, the system scale and complexity are not problems on their own, but a contributing factor to other challenge themes, such as testing, as the large scale of an ERP system requires additional testing time, and the system complexity attributes to the challenge of testing disparity.

6.3 Integration and deployment pipeline

The following integration and deployment pipeline challenges were identified: revision control and code stabilization. One challenge reported in the previous research related to establishing a functional deployment pipeline system is the low level of tooling support for Continuous Delivery, because as of yet, no standardized software solutions or products exist. This is why major global companies, such as Facebook and Google, have adopted deployment pipelines successfully by developing their own pipelines and related software tooling. (Steffens et al., 2018.) Smaller companies, however, may not have sufficient resources to achieve similar success in constructing their own pipelines from the beginning, or by integrating different development tools internally. Interestingly, the interview data seemed to contradict the fact, as there were no challenges reported with the tooling options for deployment pipelines in the interviews.

It should be noted that extremely frequent releases may not produce any additional benefits for either user organizations or the developer organization, at least not for the ERP system vendors, which means there might not be any incentive for developer organizations to aim to shorten release cycles beyond a certain point. However, there are no negative aspects in having the capability to release frequently, even if the releases actually occur scarcely. Indeed, various Continuous Integration tools or deployment pipelines would benefit even the smallest development organizations consisting of only one or few teams. Nevertheless, it might not be worth the additional effort and resources required for the smaller development teams and organizations to adopt the practices.

Integration and deployment pipeline challenges are interconnected to testing challenges at least in two ways. Firstly, most of the testing activity is actually conducted within the deployment pipeline environment and its different stages. Because of that, advanced testing activities can be seen as a precedent for more a developed deployment pipeline, meaning that subpar testing capabilities greatly hinder the development of functional deployment pipelines, eventually interfering with the adoption of Continuous Delivery. Secondly, advanced Continuous Integration processes and functional deployment pipelines might facilitate more efficient testing procedures and practices. Additionally, lack of test automation can cause a need for prolonged or additional code stabilization periods, which also hinders the capability of Continuous Delivery.

Test automation is a potential solution in order to reduce code freeze periods (Laukkanen et al., 2017b), but it might not be possible in certain situations, such as lack of resources or customer requirements, as it was the situation with Company D. Even if discontinuing freezing practices is not possible, the potential risks and challenges generated by it can be monitored and mitigated with various means, such as feature toggles, that allow uncomplete features to exist within the code while they are inactive when the software is in use (Laukkanen et al., 2017b). This was validated by the interviews also, as Company C reportedly applied feature toggles in their development in addition to the freezing practice.

An important remark regarding freezing practices is that the developers may experience increased pressure before the feature freeze periods, as they have to finish their work in time, while QA personnel experience the greatest workload during the feature freeze period, to a point that an "us and them" atmosphere between the developers and the QA personnel is established (Laukkanen et al., 2017b), affecting organizational collaboration and communication. The aforementioned situation can potentially have some implications for the division of labour, utilization of personnel, and even management of human resources, namely the developers and QA personnel. It would be reasonable to expect that while underutilization of development and QA personnel is inefficient, overburdening them is progressively more harmful, as it may lead to decreased quality of code and system failures, which will affect the overall quality of the software product negatively, effectively hindering the ability to deliver software continuously even further. However, as Laukkanen et al. (2017b) noted, sometimes freezing practice with coupled deployment and release is an essential complexity factor, and cannot be avoided. This is the case with systems that are difficult or impossible to update or upgrade undetected while they are being used, or without downtime, including mission critical enterprise systems such as ERP.

6.4 Testing

Problems and issues in testing are among the most prevalent and most critical challenges for Continuous Delivery adoption, as reported in the research literature (Laukkanen et al., 2017a; Shahin et al., 2017). Consequently, this was reportedly the situation with the companies in this study. The following two testing challenges were identified: time-consuming testing and disparity of testing.

While test automation is regarded as one of the most important factors in Continuous Delivery adoption success, not all software development organizations are able to automate all types of tests for reasons such as insufficient infrastructure for test automation, difficult process of automating manual tests, and dependencies between software and hardware (Shahin et al., 2017). This was the situation with the companies in this study also. The aforementioned reasons are at least partially caused by system design and architecture, such as unsuitable architecture for test automation, and high scale and complexity of the system, reflecting the other challenge theme of system design and architecture.

6.5 Release

The following three release challenges were identified: customer requirements and preferences, domain restrictions, and release planning. In the previous research literature, customers have been featured as a social challenge regarding continuous practices (Claps et al., 2015; Leppänen et al., 2015). Consequently, according to the interviews, the challenges caused by varying customer preferences and policies were solved by various means. Company A provided multiple options regarding the delivery method of their system. Customers can not only choose different system modules and functionalities according to their limitations, needs and preferences, but they can also choose how often the system will be updated and where it is hosted, e.g. cloud or on-premises. In addition to that, customers can select fully supported, on-demand (Software-as-a-Service, SaaS) solution with a monthly fee. This way, different customer preferences can be addressed. On the other hand, offering multiple options can increase complexity and generate additional challenges with testing and release management. It should be noted, however, that forcing a uniform and inflexible ERP solution for customers can be highly impractical and can lead to lowered customer satisfaction and retention.

Frequent releases are one of the key principles of Continuous Delivery practice. Some customers may, however, find the frequent updates undesirable for different reasons. On the other hand, other customers may welcome them or even require them in order to develop their own business operations further. This creates a situation where the system vendor has to conform to simultaneously varying, or even conflicting preferences and requirements, which is a clear challenge for any software development organization, even more so for system vendors producing mission critical and complex enterprise systems such as ERP. In fact, customer preferences and their organizational policies may interfere with Continuous Delivery (Shahin et al., 2017), which can even lead to a situation where it is impossible for an organization to adopt Continuous Delivery practices to a full extent, especially if those customers are considered vital for business operations.

However, it is important to note that release decisions are not only about preferences of the customers. Software companies are known to settle for less continuous development processes because they are more appropriate for their product and business (Leppänen et al., 2015). There are also certain other considerations, such as different legislations regarding information security management or handling of financial data, and changes in them. Some restrictions in the form of regulations and customer policies and processes can effectively prevent practicing Continuous Delivery entirely. However, as pointed out by Humble (2018), this does not mean that implementing Continuous Delivery practices even in those kind of situations would not be highly beneficial, as those policies and processes can be handled in a more efficient way by adopting Continuous Delivery practices, even if the ultimate criteria for Continuous Delivery, releasing at will, cannot be achieved.

An additional factor contributing to the release-related challenges is that not even a state-of-the-art deployment pipeline can ensure latency-free releases from deployment to production (Lehtonen et al., 2017). For example, even with a fully functional deployment pipeline in operation, some aspects of the release process, such as written user manuals, remain tedious and time-consuming, as they are difficult or even impossible to automate, at least with contemporary technologies and practices, as evidenced also by this study.

In previous research, Continuous Deployment has been regarded as a subsequent evolutionary stage of Continuous Delivery which organizations are seeking to adopt. (Olsson et al., 2012) However, none of the interviewees reported any intentions to adopt Continuous Deployment practice. As expected, it seems to be uncommon for ERP vendors to aim for practicing Continuous Deployment, even if it was technologically possible. This is first and foremost because of the critical role and large scope of an ERP system. Aside from minor changes, such as software patches, updating the system, regardless of size and scope of the update, will likely require changes in the business operations of the user organization, which in turn is most often less than ideal for the user organization, even more so if it occurs frequently.

6.6 Organizational

The following three organizational challenges were identified: coordination and collaboration, communication, and insufficient resources. Organizational challenges are not directly connected to any other challenge themes, but they encompass the entire development process of an organization, meaning that organizational challenges can impact the overall Continuous Delivery adoption of an organization.

Previous research highlights the influence of change resistance and change management (Claps et al., 2015; Leppänen et al., 2015) for Continuous Delivery adoption. Interestingly, change resistance or change management challenges were not directly reported in any of the interviews, and only in two interviews the topic was mentioned at all. However, change resistance is closely related to the coordination and collaboration, and communication challenges.

Finally, according to Olsson et al. (2012), constant changes in development tools, and the following need of learning new tools among the most critical challenges in adjusting to the practice of Continuous Integration. The interviewees in this study did discuss the tooling changes, but did not perceive them as a major challenge for adopting or practicing Continuous Integration or Continuous Delivery.

6.7 Conceptual framework of Continuous Delivery adoption challenges for ERP system vendors

The challenges of Continuous Delivery adoption can be summarized in the form of a conceptual framework as shown in Figure 5. The framework consists of the five previously discussed Continuous Delivery challenge themes including the previously identified 12 Continuous Delivery challenges. Furthermore, the declared relationships between the challenge themes are denoted with connecting arrows, with the direction of the arrow implying the direction of incluence in the relationship.

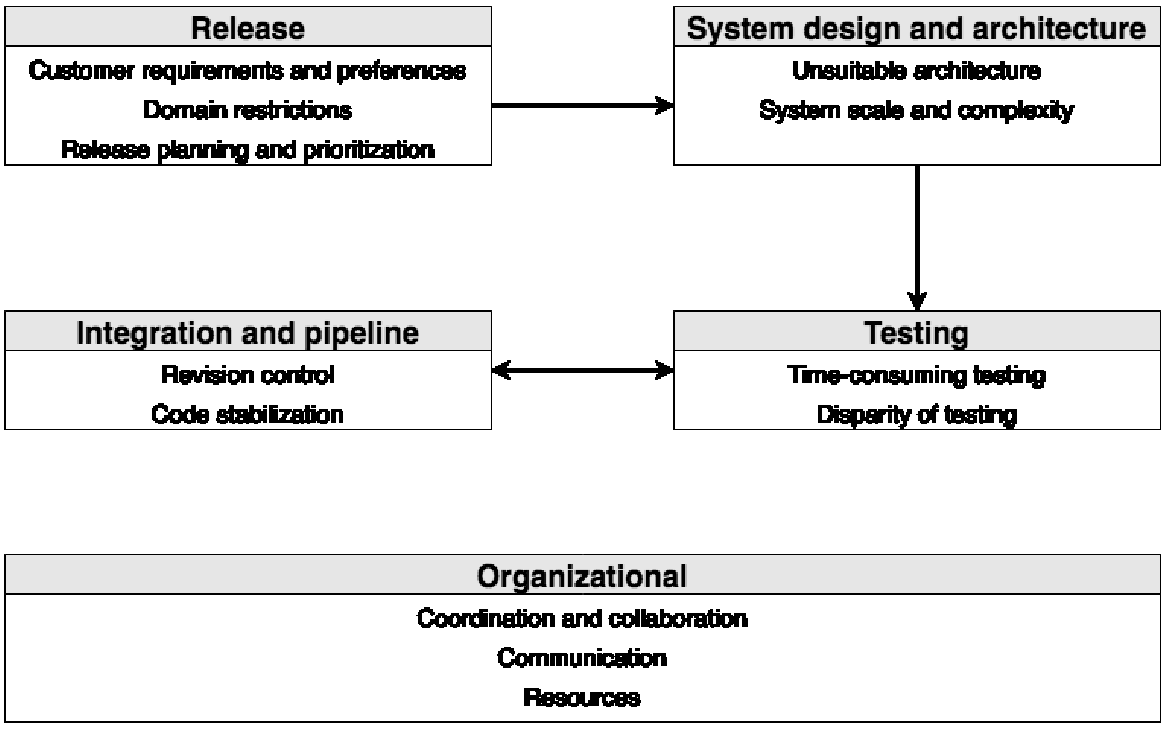

FIGURE 5 Conceptual framework of Continuous Delivery adoption challenges

System design and architecture challenges are related to testing challenges, as they are an accounting factor for testing challenges, as discussed previously. Testing challenges are related to integration and deployment pipeline challenges bidirectionally, as the testing activities are, at least partly, performed within the integration and deployment pipeline phases of the development, and the challenges in integration and deployment pipeline may account to the testing challenges. Release challenges are connected to system design and architecture challenges, as different challenges, such as domain restrictions, can affect the system architecture and complexity. Finally, while the organizational challenges are not directly connected to any other challenge theme, they encompass all other challenge themes, as organizational factors affect the entire process of software development and delivery.

7 DISCUSSION

In this chapter, the research question is addressed and practical and theoretical implications, and limitations of the study are discussed. Finally, topics for further research are considered.

7.1 Addressing the research question

The objective of this study was to identify and analyze the challenges related to adoption of Continuous Delivery practice in small to medium sized ERP system vendors. In order to identify and analyze the challenges, the following research question was presented in the introduction chapter:

- What are the Continuous Delivery adoption challenges for small and medium sized ERP system vendors?

In order to address the research question, a qualitative study was conducted, in which semi-structured theme interviews from industry experts were used as a main data collection method. The data was analyzed using a qualitative directed content analysis method, which resulted in a conceptual framework consisting of five interconnected Continuous Delivery adoption challenge themes, which include a total of 12 Continuous Delivery adoption challenges.

7.2 Theoretical implications

This study expands the body of theoretical knowledge in form of a proposed conceptual framework consisting of 12 identified Continuous Delivery adoption challenges divided into five interconnected themes in a context of ERP system development and delivery. The results of the study appear to support and confirm the existing body of knowledge rather than contradict it, which was not unexpected, as prior research literature appears to be relatively consistent with findings, partly because many research articles seem to cite only a limited amount of prior certain research articles, which are often the same. The challenge themes identified in the study were similar to those identified in previous research literature, partly because of the selected data analysis method, the qualitative directed content analysis, in which the analysis of research data is guided by existing theory or knowledge. The objective of this study, however, was not to validate or test an existing theory or knowledge, but to generate additional knowledge of the phenomenon of interest in a more limited context, the ERP system domain.

The results indicate that the challenges of Continuous Delivery adoption are similar for small and medium sized ERP system vendors compared to the general challenges presented in the literature reviews. In other words, the challenges in the ERP system domain do not appear to contradict the challenges identified in other software domains. This was somewhat unexpected, as the ERP systems are considered to be a distinct type of systems differing in multiple ways when compared to other software.

Of the challenges identified in this study, the system design and architecture challenges appeared to be the most connected to other challenges. In this sense, addressing challenges related to system architecture and design could facilitate the adoption of Continuous Delivery the most. The results of the study also indicate that while organizational Continuous Delivery adoption challenges had existed, they were already mostly solved in the companies, implying that organizational challenges are easiest to overcome. This contradicts prior research, where organizational challenges are considered critical, including change resistance (Shahin et al., 2017), which was not reported at all in the companies in this study. This contradiction may be caused by the fact that companies in this study were exclusively small or medium sized enterprises, which can potentially solve their organizational challenges more efficiently, while the organizations included in the prior research were usually large, global organizations, in which organizational changes are considerably more difficult to implement and establish.

7.3 Practical implications

Continuous Delivery practices are highly recommended even in the ERP system vendor domain, even though there are difficulties and challenges to overcome. No two organizations are completely identical, and for that reason the adoption challenges can vary depending on the organizational aspects. System vendors should identify the most critical challenges for Continuous Delivery adoption in their organization, and plan and act accordingly.

It is possible to obtain some benefits of Continuous Delivery by adopting the practice only partially, which means that vendors can select the most suitable or only the necessary methods to their software development process. For example, not all releases have to be delivered continuously. This means that certain smaller releases, such as smaller patches, can be delivered frequently and automatically, while larger system revisions and upgrades can be delivered less frequently and manually.

However, according to some industry experts, the on-premises installation delivery method should remain a prevalent delivery option for ERP systems, at least in the near future. As the on-premises installation delivery model is less suitable for Continuous Delivery than other delivery models, such as SaaS, it is not completely straightforward to decide if ERP system vendors should spend extreme resources into adopting state-of-the-art Continuous Delivery capabilities. Nevertheless, developing Continuous Delivery capabilities further will most likely affect other organizational capabilities in a positive way.

7.4 Limitations of the study

There are several major limitations in this study. Firstly, there are limitations caused by the selected data collection method. The most apparent limitation is the size of the data set, as the volume of interview data is noticeably limited, as there were only five persons representing four organizations interviewed. This was, however, somewhat expected, as it can be difficult to obtain interviewees for a thesis work. An additional limitation with the data collection method is that the interviews could not be conducted strictly according to the research guide, and furthermore, the interview guide itself was inadequately designed. Therefore, a more defined interview guide could have yielded more comprehensive and balanced data collection. As the interviews were conducted in a relatively open form, the interview themes were not covered completely proportionally between each interview, thus affecting the data collected. In addition, due to the inexperience of the researcher, the interview situations could not be controlled in a most efficient way to enable adequate data collection. Secondly, there are limitations with the selected data analysis method, the qualitative directed content analysis. The use of a theory or prior knowledge as a guidance for analysis can lead to bias, as it is more likely to find supportive evidence rather than unsupportive of a theory or prior knowledge (Hsieh & Shannon, 2005). Finally, qualitative methods in general are susceptible to research bias. In other words, the characteristics of the researcher, such as personal beliefs and views, can affect the interpretation of the data, which eventually can affect the results of the study.

7.5 Topics for further research

In order to extend the knowledge of the phenomenon of interest, and to address the aforementioned limitations of this study, further research is required. Some possible topics for further research are listed below:

- In-depth single or multiple case study of Continuous Delivery adoption challenges in ERP vendor organizations
- Large-scale survey research on Continuous Delivery adoption in ERP system vendors
- Study of critical success factors for Continuous Delivery adoption

8 CONCLUSION

The objective of this study was to identify and analyze what challenges ERP system vendors face when adopting the practice of Continuous Delivery. A total of 12 Continuous Delivery challenges were identified and classified into five separate, yet interconnected challenge themes, which were the following: system design and architecture, integration and pipeline, testing, release, and organizational. The challenges, challenge themes, and the implied relationships between challenge themes were summarized in the form of a conceptual framework

To address the research problem, the study was divided into two separate sections: a literature review of previous research, and an empirical research section. The purpose of the literature review was to obtain knowledge and understanding of the phenomenon of interest, including preceding and related software engineering practices. As the topic of research was specified to ERP system vendors, research literature related to the ERP systems was included in the literature review. The empirical research section was conducted with a qualitative research method, including semi-structured theme interviews as the main data collection method, and a data analysis method known as qualitative directed content analysis was selected as the method of data analysis. The complete research methodology was presented and discussed in chapter four.

The results of the empirical research were first presented in chapter five, and then analyzed in chapter six. The analysis of the results included comparison to prior research, with the objective of identifying contradicting or supporting evidence in relation to the identified results. Finally, in chapter seven, the research question was addressed, theoretical and practical implications were discussed, and limitations of the study and topics for further research were considered.

REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis.

Barqawi, N., Syed, K., & Mathiassen, L. (2016). Applying service-dominant logic to recurrent release of software: an action research study. *Journal of Business & Industrial Marketing*.

Basil, V. R., & Turner, A. J. (1975). Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, (4), 390-396.

Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70-77.

Cavaye, A. L. (1996). Case study research: a multi-faceted research approach for IS. *Information systems journal*, *6*(3), 227-242.

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

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

Clegg, B., & Wan, Y. (2013). Managing enterprises and ERP systems: a contingency model for the enterprization of operations. *International Journal of Operations and Production Management*, *33*(11-12), 1458-1489.

Cohen, D., Lindvall, M., & Costa, P. (2003). Agile software development. *DACS SOAR Report*, *11*, 2003.

Davis, A. M., Bersoff, E. H., & Comer, E. R. (1988). A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, *14*(10), 1453-1461.

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94-100.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review*, *14*(4), 532-550.

Elragal, A., & Haddara, M. (2012). The Future of ERP Systems: look backward before moving forward. *Procedia Technology*, *5*, 21-30.

Eurostat (2020). Structural business statistics overview. Retrieved on 29.9.2020 from https://ec.europa.eu/eurostat/statistics-explained/index.php/Structural_business_statistics_overview

Fowler, M. (2006). Continuous integration. Retrieved on 12.2.2020 from https://martinfowler.com/articles/continuousIntegration.html

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(8), 28-35.

Forrester (2019). Look Beyond ERP: Introducing The DOP. Retrieved on 28.10.2020 from https://www.forrester.com/report/Look+Beyond+ERP+Introducing+The+DOP/-/E-RES152335?objectid=RES152335

Fricker, S., & Schumacher, S. (2012, March). Release planning with feature trees: Industrial case. In *International Working Conference on Requirements Engineering: Foundation for Software Quality* (pp. 288-305). Springer, Berlin, Heidelberg.

Gartner (2020). Market Share Analysis: ERP Software, Worldwide, 2019. Retrieved on 13.1.2020 from https://www.gartner.com/en/documents/3985627/market-share-analysis-erp-software-worldwide-2019

Gholami, M. F., Daneshgar, F., Beydoun, G., & Rabhi, F. (2017). Challenges in migrating legacy software systems to the cloud—an empirical study. *Information Systems*, *67*, 100-113.

Greer, D., & Ruhe, G. (2004). Software release planning: an evolutionary and iterative approach. *Information and software technology*, *46*(4), 243-253.

Gregor, S. (2006). The nature of theory in information systems. *MIS quarterly*, 611-642.

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

Humble, J. (2018). Continuous delivery sounds great, but will it work here?. *Communications of the ACM*, *61*(4), 34-39.

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education.

Humble, J., & Molesky, J. (2011). Why enterprises must adopt devops to enable continuous delivery. *Cutter IT Journal*, *24*(8), 6.

Humble, J., Read, C., & North, D. (2006, July). The deployment production line. In *AGILE 2006 (AGILE'06)* (pp. 6-pp). IEEE.

Hsieh, H. F., & Shannon, S. E. (2005). Three approaches to qualitative content analysis. *Qualitative health research*, *15*(9), 1277-1288.

IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990.

Itkonen, J., Udd, R., Lassenius, C., & Lehtonen, T. (2016, September). Perceived Benefits of Adopting Continuous Delivery Practices. In *ESEM* (pp. 42-1).

Johansson, B., & Ruivo, P. (2013). Exploring factors for adopting ERP as SaaS. *Procedia Technology*, *9*, 94-99.

Lahtela, A., & Jäntti, M. (2011, July). Challenges and problems in release management process: A case study. In *2011 IEEE 2nd International Conference on Software Engineering and Service Science* (pp. 10-13). IEEE.

Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, *82*, 55-79.

Laukkanen, E., Paasivaara, M., Itkonen, J., Lassenius, C., & Arvonen, T. (2017, May). Towards continuous delivery by reducing the feature freeze period: a case study. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)* (pp. 23-32). IEEE.

Lehtola, L., & Kauppinen, M. (2006). Suitability of requirements prioritization methods for market-driven software product development. *Software Process: Improvement and Practice*, *11*(1), 7-19.

Lehtonen, T., Suonsyrjä, S., Kilamo, T., & Mikkonen, T. (2015, October). Defining metrics for continuous delivery and deployment pipeline. In *SPLST* (pp. 16-30).

Lenart, A. (2011, September). ERP in the Cloud–Benefits and Challenges. In *EuroSymposium on systems analysis and design* (pp. 39-50). Springer, Berlin, Heidelberg.

Leppanen, M., Makinen, S., Pagels, M., Eloranta, V. P., Itkonen, J., Mantyla, M. V., & Mannisto, T. (2015). The Highways and Country Roads to Continuous Deployment. *IEEE Software*, *2*(32), 64-72.

Maditinos, D., Chatzoudes, D., Tsairidis, C., & Theriou, G. (2011). The impact of intellectual capital on firms' market value and financial performance. *Journal of intellectual capital*.

Meyer, M. (2014). Continuous integration and its tools. *IEEE software*, *31*(3), 14-16.

Myers, M. D. (1997). Qualitative research in information systems. *MIS Quarterly*, *21*(2), 241-242.

Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and organization*, *17*(1), 2-26.

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

Nerur, S., Mahapatra, R., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, *48*(5), 72-78.

Olsson, H. H., Alahyari, H., & Bosch, J. (2012, September). Climbing the" Stairway to Heaven"--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *2012 38th euromicro conference on software engineering and advanced applications* (pp. 392-399). IEEE.

Panorama Consulting Group (2019). 2019 ERP Report. Retrieved on 14.1.2020 from https://www.panorama-consulting.com/resource-center/erp-software-research-and-reports/panorama-consulting-solutions-2019-erp-report/

Rajkumar, M., Pole, A. K., Adige, V. S., & Mahanta, P. (2016, April). DevOps culture and its impact on cloud delivery and software development. In *2016 International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Spring)* (pp. 1-6). IEEE.

Riungu-Kalliosaari, L., Mäkinen, S., Lwakatare, L. E., Tiihonen, J., & Männistö, T. (2016, November). DevOps adoption benefits and challenges in practice: a case study. In *International Conference on Product-Focused Software Process Improvement* (pp. 590-597). Springer, Cham.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L. E., Teppola, S., Suomalainen, T., Eskeli, J., ... & Oivo, M. (2017). Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, *123*, 263-291.

Royce, W.W., (1970). Managing the development of large software systems: concepts and techniques. 1970 WESCON technical papers. Vol. 14. 723.

Sarker, S., Xiao, X., & Beaulieu, T. (2013). Guest editorial: qualitative studies in information systems: a critical review and some guiding principles. *MIS Quarterly*, *37*(4), iii-xviii.

Shang, S., & Seddon, P. B. (2000). A comprehensive framework for classifying the benefits of ERP systems. *AMCIS 2000 proceedings*, 39.

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

Shahin, M., Zahedi, M., Babar, M. A., & Zhu, L. (2019). An empirical study of architecting for continuous delivery and deployment. *Empirical Software Engineering*, *24*(3), 1061-1108.

Sneed, H. M. (2006, March). Integrating legacy software into a service oriented architecture. In *Conference on Software Maintenance and Reengineering (CSMR'06)* (pp. 11-pp). IEEE.

Sommerville, I. (1996). Software process models. *ACM computing surveys (CSUR)*, *28*(1), 269-271.

Steffens, A., Lichter, H., & Döring, J. S. (2018, May). Designing a next-generation continuous software delivery system: Concepts and architecture. In *2018 IEEE/ACM 4th International Workshop on Rapid Continuous Software Engineering (RCoSE)* (pp. 1-7). IEEE.

Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S. B., & Shafique, M. U. (2010). A systematic review on strategic release planning models. *Information and software technology*, *52*(3), 237-248.

Ståhl, D., & Bosch, J. (2014). Modeling continuous integration practice differences in industry software development. *Journal of Systems and Software*, *87*, 48-59.

Umble, E. J., Haft, R. R., & Umble, M. M. (2003). Enterprise resource planning: Implementation procedures and critical success factors. *European journal of operational research*, *146*(2), 241-257.

Waller, J., Ehmke, N. C., & Hasselbring, W. (2015). Including performance benchmarks into continuous integration to enable DevOps. *ACM SIGSOFT Software Engineering Notes*, *40*(2), 1-4.

Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing*, *30*(3), 32-39.

APPENDIX 1 INTERVIEW GUIDE

Initial version

Theme	Examples of discussion prompts
Background information	Description of the company, role in the company
Release management	Challenges
Strategic release planning	Challenges
Testing, deployment & support	Testing activities, types of testing, challenges
Release scheduling	How releases are scheduled, chal-lenges
Release content	Challenges

Revised version

Theme	
Build design	
System design	
Integration	
Testing	
Release	
Human and organizational	
Resources	

APPENDIX 2 INTERVIEW QUOTATION TRANSLATIONS

1 ”Siihen on prosesseja, miten me sitä testausta tehdään, mutta me ei oo vielä ehditty ihan hirveesti tehdä niinkun järkeviä automaattitestejä. Joihinkin tiettyihin juttuihin niitä löytyy, mutta ei sillain isossa laajuudessa.” (B1)

2 “Nyt ei suunnitella niinkun, että mitä tehdään tällä tietyllä aikavälillä, vaan suunnitellaan – puhutaan niinkun featuresta, että jos kehitetään jotain uutta featurea, niin suunnitellaan tavallaan sen tekeminen, että – ja se on pätkitty semmoisiin pienempiin kokonaisuuksiin ja tavallaan niinkun suunnitellaan, mietitään että paljonko sen featuren tekeminen vie aikaa ja milloin se aloitetaan, että meillä ei silleen niinkun mitään viikon tai kahden sprinttejä oo vaan ne sprintit tapahtuu tavallaan niiden featureiden sisällä --- Ne voi olla niinkun viikkoa tai kahta, tai mitä onkaan, mutta tavallaan suunnitellaan sieltä niinkun lopputuloksen näkökulmasta se.” (D1)

3 ”Täällä jos on kymmenen hengen tiimi niin nehän siirtelee vaikka käsin ne, käyt jokaiselta erikseen kysymään, että missä vaiheessa tuo ticket on ja, et käsin siirtää sen. Mutta jos aattelee että olis vaikka 100 koodaria, niin siinä onkin vähän enemmän haastetta pitää niinkun ajan tasalla se systeemi.” (C1)

4 “Tää meidän vanhin tää tämmönen, aletaan jo puhua ehkä vähän legacystakin järjestelmästä mikä sitten toimittaa tätä meidän ERPin virkaa siellä ja on tämä tavallaan niinku kaiken meidän tekemisen keskiö. --- Ja sitten tän toiminnanohjauksen ympärille meillä on alkanut tässä vuosien varrella alkanut tulla kaikenlaisia liitännäistuotteita.” (B1)

5 “Jos sä saat jonkun isomman kokonaisuuden tehtyä, niin se järjestelmä alkaa toisesta päästä vanhenee, se tekniikka --- Mutta se on vaan semmonen asia mikä pitää hyväksyä ja kun tämä on sama järjestelmä, mitä on alettu kehittämään yli 10 vuotta sitten, että kehitetään sitä samaa järjestelmää edelleen, niin se on monelta osin [legacya], siellä on vanhaa tapaa tehdä asioita.” (C1)

6 “Tekee tämmöstä ERP-järjestelmää, mikä on niinkun hirveen monipuolinen ja laaja ja se, että tänne tehdään jatkuvasti muutoksia niin se että niitä virheitä ei syntyisi ollenkaan. Että niitä yhdistelmiä, mitä pitää testata eri kieliä, eri alustoja varsinkin nyt kun meillä on tää onlineversio niin niitä on niinkun niin tuhoton määrä, niitä käyttötapoja ja reittejä on tuhoton määrä.” (A1)

7 “Se päivitys on pikkusen erilainen kun johonkin vaikka kuluttajatuotteeseen --- Meillä se päivitys tarkoittaa monesti sitä, että se vaatii koulutuksen, se vaatii meillä projektipäällikön niinkun käymisen siellä, et miten tää vaikuttaa teidän liiketoimintaan --- Ne on yleensä ihan maksullisia juttuja, koska ne vaikuttaa

monella tapaa yleensä siihen niinku asiakasyrityksen liiketoimintaan myös.” (B1)

8 “Me oikeestaan luovuttiin tästä master-haarasta, meillä on nykyään tää integration-haara pelkästään olemassa mihinkä me koodaillaan muutoksia.” (A1)

9 “Meillä on sitten semmosia keissejä, tää meidän tavallaan tuotteen luonne, kun se on toiminnanohjausjärjestelmä ja käyttöönotot on tosi pitkiä monesti asiakkailla, ne voi kestää vuodenkin. Kun tuotteen käyttöönotto siellä ei välttämättä oo järkevää mennä sillä asiakkaalla aina tietyn versiotägin mukaan” (B1)

10 “Viikkoa ennen tehdään content freeze siitä, eli silloin ei saa enää laittaa uusia, tai uutta koodia ei saa, että korjataan sitä niinkun mikä siellä on jo sisällä, että jos siihen liittyen testauksessa löytyy jotakin, että saadaan pidettyä se julkaisuaikataulu, että se on aina tiettynä päivänä.” (C1)

11 “Me jäädytetään se versio jossain vaiheessa ennen kuin sitä ruvetaan tota, tekemään sitä versiota ulos. Eli jäädytyksen aikana koodareilla on ns. blokki siihen, että mitään mitä ne tekee, niin niiden PR:t, niin niitä ei liitetä ainakaan masterhaaraan.” (D2)

12 “Et meillä on tavallaan se automatiikka ja kaikki on ihan valmista, ettei meidän olisi pakko kenenkään koskee tonne mihinkään vaiheeseen että nää paketit lähtis asiakkaille julkaisuun asti esim. meidän omilla alustoilla. Mutta me ollaan haluttu sinne rakentaa semmoset stopit että siellä on pieniä semmosia tsekkauksia, että kaikki tavallaan on semmonen hengähdystauko, että voi vielä varmistaa ennen kuin paketti lähtee eteenpäin jaeltavaksi.” (A1)

13 ” Ei me voida kaikkia tota, tai kaikkea mitä se meidän softa tekee niin testata mitenkään, ihan sama vaikka me oltais automaatio- tai tehty kaikkiin asioihin automaatiotestit, niiden ajamisessahan menis kuukausi.” (D2)

14 “Kattavuus haetaan sillä, että etitään ne ns. blockerit, eli jos- tai saadaan sellaiset featuret, mitkä on pakko toimia, eli joku tiedon tallennus, asiakkaan lisääminen, tällaiset niinkun et jos nää ei toimi, niin sitä versiota ei voi julkaista.” (D2)

15 “Vaikka saatais meidän end-to-end tai acceptance testit mimmoiseen kuntoon tahansa, niin varmaan ilman manuaalista testausta ei pystytä tai ei uskalleta koskaan lähteä niinkuin tekemään deliveryjä, että tuota varmaan siinä jonkinlainen manuaalinen testaus tulee joka tapauksessa pysymään kuvioissa mukana, että silleen se ehkä siihen kuvioon vaikuttaa, että jos tekis jotain simppeliä puhelinappia tai jotain muuta, niin siinä ei varmaan ois kauheesti se testaus, olis ratkaistavissa ihan kokonaan automaatiolla suurin piirtein.” (D1)

16 ”Tavallaan ne variaatiot mitä siinä testauksessa pitäis huomioida niin se niinku tuntuu olevan vähän sellainen melkein ylitsepääsemättömän iso juttu.” (A1)

17 ”Haasteena on se, että on niin erilaisia järjestelmiä, mitä tohon meidän liiketoiminta-alustaan liittyy, ihan teknologisesti, mutta myös tavallaan luonteeltaan erilaisia, että tota on niinkun legacympaa Windows-työpöytä niinkun käyttöliittymää, minkä testaaminen on taas niinkun automatisoidusti huomattavasti erilaista kuin vaikka jonkun oikeesti APIen läpi menevän niinkun rajapintatoiminnallisuuden testaaminen, että nää on niinkun aivan täysin ääripäitä testausmielessä. Niin tota se on ehkä se haaste, että niitä niinkun käyttötapauksia on niin paljon erilaisia ja eri tyyppisiä.” (B1)

18 ”Tää niinku tää aiheuttaa sille testaukselle hirveet haasteet, varsinkin jos puhutaan automaatiotestauksesta, koska se vaatis meille varmaan tänne semmosen 200 henkilön automaatiotestausporukan, joka niitä testejä ylläpitää ja koko ajan niinkun synkkaa projektien kanssa, että sen sais niinkun toimimaan, niin siinä ei oo mitään järkeä tavallaan.” (B1)

19 ”Mun linjaus on nyt ollut se, että niinkun tietyt semmoset perustoiminnallisuudet, mitkä pysyy suurin piirtein muuttumattomana versiosta toiseen, niin saatais sen automaatiotestin piiriin --- Näitä kaikkia meidän parametrisointivaihtoehtoja ja mitä kaikkea nyt tuleekaan mieleen käyttötapausta, niin ei millään pystytä testaamaan eikä millään pystytä niinkun varsinkaan automatisoimaan. Että tää on se haaste.” (B1)

20 ”Kun järjestelmä kehittyy eteenpäin, niin testicasejen pitää myös elää. Eli jokainen testicase- tää on se ongelma, niinkun minkä takia automaattitestit on tosi raskaita ja niitä on tota vähän hankala tehdäkin, kun siitä tulee sellainen, yllättävän nopeesti tulee sellainen ylläpitohelvetti. Ja jos niitä ei ylläpidetä, niinkun sanoit, ne testaa väärää asiaa. Jonka jälkeen se on ihan sama, että onko se testi validi vai ei.” (D2)

21 ”Ehkä se on justiin se miten paljon nämä vaatimukset on niinku asiakkailta nousseet ja miten tää teknologia kehittyy koko aika.” (A1)

22 ”Monesti ihan niinkun meidän kontrakti- tai niinkun mikä se on suomeksi- sopimukset niinkun kieltää tiettyjen tahojen kanssa, että heille ei saa toimittaa jatkuvasti vaan pitää mennä niinkun oman asiakastestauksen kautta ensin.” (D1)

23 ”Yksinkertasta jos ajattelee devauksen näkökulmasta, että me julkaistaan paketteja koko ajan, mut sit kun taas tulee jotakin tämmösiä isompia tai merkittävämpiä asioita niin kyllä sitä niinkun pohditaan koko aika että millon niitä olis tarpeen saada pihalle, että kyllä meiltä löytyy ihan niinkun roadmappi mi-

hin on aikataulutettu näitä isompia kehitysasioita ja sinne on mietitty sitten niinku priorisoitu.” (A1)

24 “Alunperinhän se oli ohjelmoijille semmonen että vähän niinkuin negatiivisen tuntuinen olo, että hirveesti kytätään tekemistä, että eikö täs niinku luoteta meihin, että me tehdään kunnolla hommia.” (A1)

25 “Omasta näkövinkkelistä niinkun suurimmat tämmöset henkilöstöön ja organisoitumiseen liittyvät haasteet on vähäsen jo selätetty, kun on saatu nyt vihdoin ja viimein kaikki nää meidän tiimit toimimaan saman sprinttisyklin mukaisesti ja kaikille on saatu tavallaan se sprinttiajatusmaailma tonne niinkun päivittäiseen tekemiseen mukaan” (B1)

26 “Tää on niinkun se meidän homma, että tarvii koko ajan olla hirveen skeptinen siihen, että asiakas tietää ja ymmärtää, mitä sen pitää tehdä. Ja se heijastuu meille sitten siihen, että pitää ensin testaa se silleen kuin koodari haluaa, sen jälkeen testata se silleen miten me halutaan, ja sen jälkeen koitetaan rikkoa se silleen, miten asiakas sitä käyttäis.” (D2)

27 “Tää on ollut yllättävän iso haaste saada tämmönen niinkun iso tuotekehitysporukka puhumaan samaa kieltä.” (B1)

28 “Nehän on niinkuin tavallaan työkaluja, jotka tekee siitä Continuous Integrationin ja Deliveryn, tai Integration on tavallaan se, että mikä tekee siitä jatkuvasta, tai pitäis tehdä siitä jatkuvasta julkaisemisesta turvallista, että tuota eihän se ihan aina niin mene, mutta se on niinkun välttämättömyys siihen, että voidaan julkaista ylipäätään tai voidaan julkaista nopeasti.” (D1)