From Traditional Development to Continuous Deployment: Strategies and Practices in CI/CD Pipelines

Niko Benkler

Supervisor: Dr. Robert Heinrich
robert.heinrich@kit.edu

Abstract. In the last decade, software development experienced a huge transition. Since agile methodologies were introduced in the early 2000, software development became significantly faster. Today, another software development process is emerging: Continuous Software Engineering (CSE). CSE, especially Continuous Integration(CI), Continuous Delivery(CDE) and Continuous Deployment(CD) receive more and more attention in organizations such as Facebook, Paddy Power and Atlassian, but also in small start-up companies. It enables them to e.g. deliver software more frequently, reduce time-to-market, obtain customer feedback faster, create the right product or to improve product quality. This seminar paper aims to present the current state of the art concerning Continuous Practices, compare the traditional deployment strategies with the new CSE practices and proposes tools, that can be used in a CI/CD Pipeline to support CSE. Also, it reveals the barriers an organization has to face while adopting Continuous Practices. CSE not only comes with benefits, it also has negative side effects, such as a time consuming and costly implementation or an increasing mental stress for development team members.

Keywords - Agile, continuous software development, continuous integration, continuous delivery, continuous deployment, DevOps, CI/CD Pipeline

Table of Contents

1 Introduction

Today, the software development process has to face many difficult demands. The area of fast-changing and unpredictable markets, changing customer demands and rapidly advancing information technologies [OlAB12] [ClSA15] require a faster and more flexible process of software development. To achieve this, several organizations adopt Continuous Practices in order to extend their agile practices [ClSA15] or to replace their traditional development methodologies. Therefore, releasing software becomes even faster.
This seminar paper presents a possible evolution path, referenced as 'stairway to heaven' [OlAB12], which describes a transition from traditional development towards CD. The core of CSE is a deployment pipeline [SCLZ+16]. Therefore the paper also presents its usual phases and the possible tools, that support each phase. Nevertheless, several studies showed, that CSE not only comes with benefits, but also with huge social and technical challenges [ClSA15]. The work clarifies them and proposes some mitigation strategies. The remainder of the seminar paper is structured as follows: In section 2, the terminology is defined. Section 3 describes a possible transition from traditional deployment to CD, based on the 'stairway to heaven' model [OlAB12] [SBZZ17].This includes several deployment strategies used to adapt CSE practices and the comparison between traditional deployment and the CD process. That section is followed by an explanation of a possible pipeline and the available tools that can be used to support the tasks of each phase. Section 5 discusses the challenges which occur when adopting CSE practices. Finally, the conclusion is presented in section 6.

2 Background

Here,the overview of the most important keywords is given. Those are necessary to understand the content of the seminar paper. When studying the available information about Continuous Software Engineering (CSE), one could clearly see that there are no universally accepted definitions [SCLZ+16]. Terms like CDE and CD are sometimes used interchangeable, albeit there is a small but relevant difference [ShBZ17]. Therefore, the following subsections provide the definitions as they are used in this seminar paper.

2.1 Agile Methodologies

Agile software development is based on "iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams" [TaKB16]. It encourages adaptive planning, early delivery and iterative improvement. Besides, other characteristics such as flexibility, efficiency, speed, the ability to react to fast changing customer demands and fluctuating market needs [OlAB12] make agile methodologies so attractive for organizations.
As mentioned before, agile methodologies were introduced in the early 2000s

[ShBZ17]. So far, many development companies succeeded in implementing agile methodologies. However, agile software development 'only' allows frequent software release, not continuous release. This difference becomes clear, when it comes to the comparison of traditional deployment and CD. The original "Manifesto for agile software development" popularized by Kent Beck did not include any automation.

2.2 DevOps

"**DevOps** is a set of practices intended to reduce the time between committing a change to a system and being placed into production, while ensuring high quality" [BaWZ15]. It is a coined word, that stands for Development (Dev) and Operations (Ops). The main task is bringing together programmers, testers and quality assurance engineers but also IT operations staff [ClSA15] primarily to shorten feedback loops. According to [BaWZ15], DevOps practices can be differentiated in 5 categories:

- Treat Ops as "first-class citizens" by involving them into the development process. Do not exclude them from development.
- Dev has to be more responsible for incident handling as in traditional deployment to shorten the time between error observation an the error fix.
- Include both, Dev and Ops into the deployment process in order to avoid errors caused by buggy deployment.
- Use of continuous deployment.
- Use of deployment scripts and other infrastructure code. This code is subjected to the same quality control practices as the application code. As a result, deployment errors due to misconfiguration can be mitigated.

Especially the fourth category shows the close connection between DevOps practices and continuous practices.

2.3 Continuous Software Engineering

Continuous software engineering **(CSE)** is a practice, that promotes the development, deployment and the release of software products in very short cycles, typically hours or days [ShBZ17] [TaKB16]. As a direct consequence, quick feedback promotes i.a. i) the determination of new functionality that should be developed, ii) feature prioritization, iii) the collection of information about the suitability of the current software architecture, iv) the collection of data for decision making in general. CES requires agile methodologies and DevOps practices [TaKB16]. According to [ShBZ17], CES involves 3 phases: Business Strategy and Planning, Development, Operations. This seminar paper focuses on three development activities which belong to the second phase (development): continuous integration, continuous delivery and continuous deployment. Fig. 1 demonstrates the relationship between CI, CDE and CD.

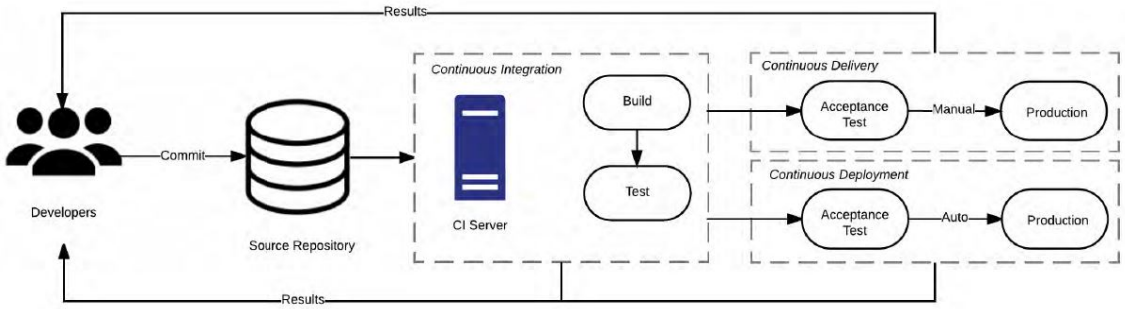

Fig. 1. The relationship between CI, CDE, CD [ShBZ17]

2.3.1 Continuous Integration Continuous Integration **(CI)** is a CSE practice that focuses on frequently integrating and merging produced code into a shared repository (e.g. multiple times a day) [ShBZ17] [SCLZ+16]. Automated builds and tests also belong to the idea of CI. Therefore, build and test errors can be identified much earlier. CI improves effectiveness of a team as well as software quality [ShBZ17]. In other words, CI ensures, that the software is always in a ready to deploy state [ClSA15].

2.3.2 Continuous Delivery Continuous Delivery **(CDE)** is a software engineering concept ensuring that the produced software can be released to production at any time. Preconditions for CDE are CI and a mechanism (e.g a tool) to automatically deploy and deliver software to a production like environment [ShBZ17]. This practice has several benefits, including a less risky release process [LLIP+16] due to automation and the ability to learn from real usage data [OlAB12]. As a consequence, the software developers can decide whether it is worth continuing the work on a certain feature or discard it. As shown in Fig.1, the release process is manual. This is concurrent with the definition of CDE, where it is mentioned that software 'can' be released automatically instead of 'is' released. Therefore, the CDE practice is classified as a pull-based approach, "for which business decides what and when to deploy" [ShBZ17].

2.3.3 Continuous Deployment An extreme version of CDE is Continuous Deployment **(CD)**, where software 'is' actually released automatically. The goal of CD is to frequently deploy the software in a production environment. This happens for example several times a day (after each commit), once a day (nightly build) or several times a week (weekend builds). Fig. 2 provides a quick overview of CDE and consequently CD benefits. CD is a push-based approach [ShBZ17], as no manual task takes place in order to deploy the software to customer site. The CD (and CDE) practice is attained by using a deployment pipeline [LLIP+16], which is explained in Section IV. Whereas CD implies the usage of CDE prac-

tices, the inversion is not correct [ShBZ17]. Due to barriers such as shortage of automated (user) acceptance tests, poor automated test coverage [SBZZ17] or the unsuitability of software types like embedded systems [ClSA15], a forced transition from CDE to CD might not always be a good idea. Furthermore, deploying software is sometimes subjected to business decisions. This topic is discussed in Section V.

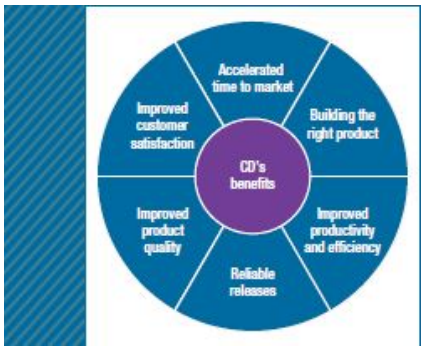

Fig. 2. CD and CDE Benefits [Chen15a]

3 The evolution path towards Continuous Deployment

This section describes the transition from traditional development to Continuous Deployment and some main practices associated to these phases(Fig. 3). Many software companies are subjected to constant alteration concerning their software development strategy in order to stay competitive. Therefore, they need to improve their way of building software.
It is important to note that the application model plays a big role for the success of the CD adoption. Albeit the traditional 'waterfall model' may still be adequate for some companies offering software for embedded systems, CD gains more and more attraction for Web-based applications [SCLZ+16]. In the remainder of this section each phase of the "stairway to heaven" is discussed in detail.

3.1 The Stairway to Heaven

The main ideas within this section originate from [ShBZ17], [SCLZ+16] and [OlAB12]. They refer to CD as a practice to keep the product "perpetually in a shippable state" using a "fully or at least partially automated deployment pipeline" [SCLZ+16]. This shows the interchangeability of CDE and CD as they are combining CDE and CD in one term. Besides, they put another phase on top of CD: "Partial Rollouts" (Fig.3). [SCLZ+16] also points out that the adoption of CD does not always has to follow the "stairway to heaven". For example,

[TaKB16] presents the "Rugby Process" approach which also adopts CD, but in a different way.

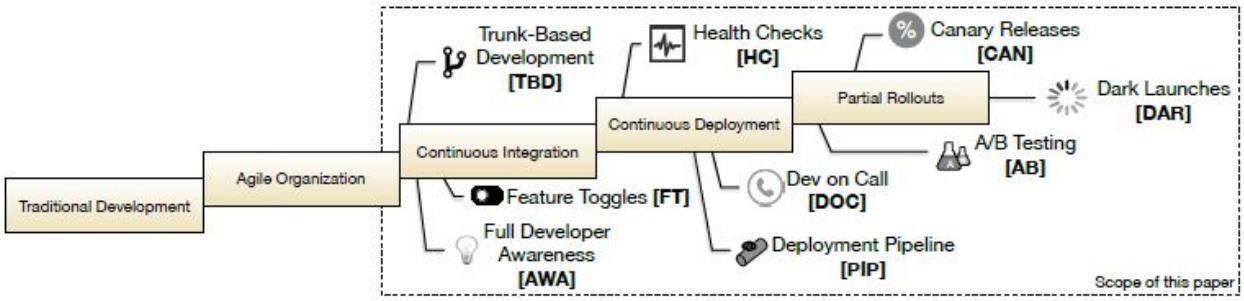

Fig. 3. The "stairway to heaven", and the main practices associated to the phase in this evolution step[SCLZ+16]

3.1.1 Phase 1: Traditional Development Traditional development is an approach, that is characterized by slow development cycles, the "waterfall model" and a bad customer integration [OlAB12]. The delivery and deployment process typically stands at the end of the project. Usually, the developed software is formed as a monolithic 3-Layer architecture with all the disadvantages that come along. It has to be noticed, that it is not the aim of this seminar paper to evaluate the suitability of this kind of software development and architecture. For successfully implementing Continuous Practices, it is simply not suitable [ClSA15] [ShBZ17] [ShBZ16] [SBZZ17]. The next step on the evolution path from Traditional Development towards Continuous Deployment is to adopt Agile Methodologies.

3.1.2 Phase 2: Agile Organization A brief introduction in Agile Methodologies is given in section 2 . Agile Organization, of course, includes agile methodologies, but "product management and system verification still work according to the traditional development model" [OlAB12]. To successfully proceed to an agile organization, the most important aspect is mindset. The resistance to change is a challenge for each evolution step. Therefore, managerial support is needed to convince each employee of the initiative. Having a few agile teams (pilot teams) promoting their methods within the organization, has a positive effect on the whole company [OlAB12]. Besides, there is a strong need to improve information exchange within and across the teams. As a result, the development teams are organized in small and cross-functional teams. The members of cross-functional teams cover all aspects of software development and delivery [TaKB16]. This enables the organization to have feature teams instead of component teams. Feature teams design and develop the entire features. This supports iterative

development. As a consequence, the release frequency improves. It has to be noted, that the development teams do not yet continuously integrate their code to an organization-wide shared code base. So far, they develop a feature within their team and as soon as it passes all required test, they merge it into the main project.
The next step of the "stairway to heaven" is Continuous Integration. The following subsection provides some basic concepts to achieve CI.

3.1.3 Phase 3: Continuous Integration The core of CI is the continuous integration of code into a shared repository across all teams while developing a software product. Again, a cultural shift is necessary in order to implement CI practices [OlAB12]. Branching strategies like trunk-based development (TBD), the implementation of feature toggle (FT) or the concept of full developer awareness (AWA) facilitate practising CI.
The continuous integration of code by many different teams can result in long-living parallel development branches. To reduce complexity, [SCLZ+16] proposes trunk-based development, where all teams integrate into one single branch, called master branch or master trunk.

Features that are not ready to be deployed, but already integrated into master need to be managed, as they should not affect the current software. This can be done by using Feature Toggles. In very simple terms, Feature Toggles permit to switch on and off a certain code block. Whenever a feature is ready for production, it gets activated. Another concept to enable and promote full transparency of the project (e.g build status, test coverage, latest client version or status and progress of features) is called (full) Developer Awareness(AWA). AWA is realized by dashboards, public monitors in the office or chat tools. This simplifies the information flow and creates transparency.
Fig. 1 demonstrates, that not only code integration, but also automated builds and test play a huge role in Continuous Practises, especially CI. Adopting "test-driven development" (TDD), "test planning" and "cross-team testing" improves the test phase during CI. TDD is a process in which requirements or user stories are turned into very specific tests. It is divided into 4 different steps: First, add a test based on a required feature; second, run all tests and see which test fails; then write or refactor the code; finally, run the tests again. Repeat those steps in very short development cycles until all test cases are passing. This helps to continuously check the quality of the written code. "Test planning" is a concept to intensify the collaboration between Quality Assurance (QA) and developers according to the DevOps principles. Together, they produce a comprehensive list of automated tests. This "liberates QAs from manually testing the majority of the software application for regression bugs" [ShBZ17]. In "cross-team testing", tests are performed by a group of developers from another team and vice versa. Besides, defining new roles like a build sheriff is a useful practice to promote CI's success. The build sheriff constantly watches the CI process or indicates that build process failed [ShBZ17].

3.1.4 Phase 4: Continuous Delivery and Deployment As mentioned in the introduction of this section, CD is a practice to keep the product "perpetually in a shippable state" using a "fully (CD) or at least partially(CDE) automated deployment pipeline" [SCLZ+16]. The core practice of CD, the deployment pipeline, is discussed in section 4. CD requires not only the development and operations team (DevOps), but also the product management as an interface between developer and the customer. [OlAB12]. [ShBZ17] claim that work structures have to be clarified. There is a need of defining new roles and teams when adopting CD, such as a release coordinator or release manager. Fig. 4 demonstrates a potential team structure as it can be found in an software developing organization using CSE for their multiple projects.

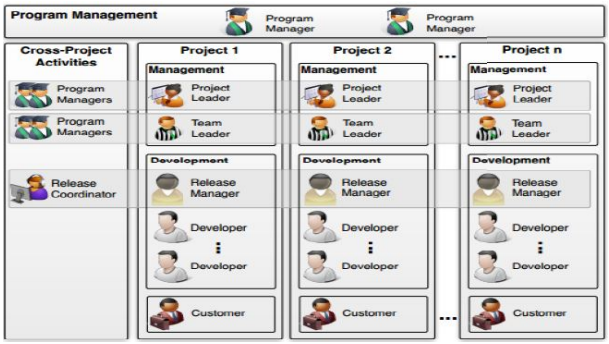

Fig. 4. Possible team structure in a CSE Team [TaKB16]

Further practices to successfully implement CD are "health checks" (HC) and the idea of "developer on call"(DOC). Monitoring on infrastructure level (e.g CPU utilization) and application level (e.g response time) is used to eliminate 'health' problems of the deployed software. Those "health checks" can also be used to support rollout decisions (e.g increase traffic for a canary release) [SCLZ+16]. "Developer on call" is associated with the idea of DevOps: Developers should be available and responsible for their code, all the more after deployment. In case of any occurring error after a code change, they know best where to find and how to fix it.

3.1.5 Phase 5: Partial Rollouts [SCLZ+16] describes partial rollout practices as "the epitome" of CD. Releasing software is always the critical part in software engineering, as customers may receive buggy code. To mitigate this problem, [ShBZ17] and [SCLZ+16] propose the use of "canary release", "A/B Testing"(AB) or "Dark Launches"(DAR). Those practices are considered as "run-time quality assurance and requirements validation techniques" [SCLZ+16]. Canary Release is a deployment method where a new software version is deployed only to a "small set of users". This mitigates the influence of buggy code and at

the same time, produces some real usage data. With that, a team decides if either the released version needs further improvement or can be released completely. A/B Testing is similar to canary release. It requires a bigger user base, as there are two versions released parallel. By collecting statistics, the team is able to decide whether version A or B performs better. Dark Launching is the process of deploying functionality to production in order to test it, but with limited or no access for any users. This enables the team to test new functionality in the production environment and consequently to decide, whether the feature is ready to "handle production-scale traffic" [SCLZ+16]. To dark launch a feature, the team implements the old and the new feature with Feature Toggles and decides which one to show in the user interface.

3.2 A comparison between Traditional Development, Agile Development and the CD Process

The following table compares the development and deployment of the traditional, agile and CD process based on [ClSA15].

	Traditional development	**Agile development**	**Continuous Practices**
Team size	Big teams, a few dozen people	Small teams, 2-8 people	Small teams, 2-8 people
Team competence	Divided in discipline	Cross-functional teams	Cross-functional teams and new roles
Planning	Rigorous planning phase at the beginning	Iterative planning in short cycles	Continuous planning in short cycles
Testing	Explicit testing at the end of the project	Testing at the end of each cycle	(Semi-)automated tests with every pipeline run
Architecture	3-Layer, monolithic, strong dependencies between modules	3-Layer or Microservices, less dependencies	Aim: almost no dependencies, independent deployability of modules
Frequency of deployment	Once waterfall process finished (months, year)	Weekly	Daily
Relative risk of deployment	High: no automation, infrequent process, huge amount of code	Still High: no automation but frequent process	Low: automation, small amount of changes, only few dependencies
Customer/ developer feedback loop	Long due to infrequent deployments	Depends on the deployment frequency	Very short, customers constantly give feedback
Unnecessary features	High amount due to long feedback loop	moderate feature rejection	prevents development of any wasted software

4 Deployment Pipeline and Tools

A Deployment Pipeline is considered to be the core of the CSE process. It enables organizations to automate code integration, build process, testing, configuration and deployment. Common stages of CD/CDE pipelines are "version control", "build", "continuous integration","artifact repository management", "unit/integration/acceptance/performance testing", "deployment", "configuration and provisioning", "log management and monitoring", "containerization" or "issue tracking" [SBZZ17] [TaKB16]. The vast amount of possible stages demonstrates, that there is no universally applicable pipeline. [Chen15a] goes even further: Pipelines should be redesigned for each application to best suit the needs.
Nevertheless, the main idea of a pipeline in CDE is always the same: Moving from stage to stage is (at least semi-)automated. If any error occurs, the pipeline stops and notifies the development team. They fix it and start the pipeline again by committing the changes. Thus, the engineering process of a pipeline, the software's architecture, the degree of automation or the available tools generate serious challenges. These are discussed in Section 5.

4.1 Example: A Deployment Pipeline

The example pipeline is divided in 7 stages: 1. Version Control System, 2. Code Management and Analysis, 3. Build System, 4. Continuous Integration Server, 5. Testing, 6. Configuration and Provisioning, 7. Continuous Delivery/Deployment Server [ShBZ17].

Fig. 5. Deployment Pipeline based on [ShBZ17]

The **Version Control System** manages the continuous code push to the repository. It tracks files to create a history of the development process. The most popular examples are GitHub and Subversion. GitHub Issues as an Issue Tracking software can be integrated here as well. [TaKB16].
Code Management and Analysis tools check the code concerning static quality metrics. Those can be: code and test coverage, coding standards like comments, alignment or duplicated code. SonarQube is one of the most popular tools, as it can be fully integrated in CI environment [ShBZ17].
Build System tools simplify the build process. They compile the source code automatically into binary code and packages. Popular Tools are Apache Maven or Ant.
The **CI server** observes the code repository. According to the respective build

policy, it triggers the Build System and runs the tests whenever a change is committed. Bamboo, Jenkins and Travis CI are the most common CI Server tools.
Testing is the key aspect to guarantee high quality software. Due to this there are various different kinds of tests: Unit tests, regression tests, (user) acceptance tests, integration tests or performance tests. JUnit is a common tool to run unit tests. To store the results of acceptance tests, [ShBZ17] proposes the test management framework TestLink. Besides, the CRTS(Continuous Regression Test Selection) and CTSP(Continuous Test Suite Prioritization) approaches intend to support effective regression tests within the CI server environment. The combination of CUTS and CruiseControl (CI Server tool) is used to continuously run system integration tests.
Automating the **Configuration and Provision** of servers, test environments, virtual machines or infrastructure is an innovation in deployment pipelines [ShBZ17]. Usually, a vast amount of time is needed to set up those environments [Chen15b]. Therefore, automation is absolutely necessary but there are currently not that many tools available.
The **CD Server** manages the final stage in Continuous Software Engineering which is the release process. Having a tool that automatically deploys code to production (either internally or externally) is substantial for successfully accomplish the goal of CSE. HockeyApp is a well-established CDE server. Web Deploy or Deployr are software tools to automatically deploy code to production. [ShBZ17].

4.2 Tools in a Deployment Pipeline

Fig.6 shows a selection of tools that can be used to build a pipeline. They are split into the stages of the pipeline. Some of them are expensive enterprise tools, others are Open Source. This seminar paper does not aim to give a detailed comparison of those tools, as this goes beyond the scope of this work.

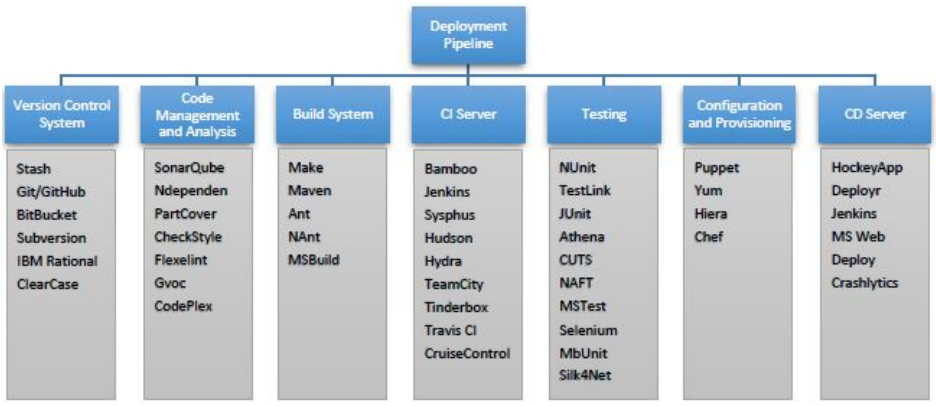

Fig. 6. An overview of tools used to form a deployment pipeline [ShBZ17]

5 Challenges for adopting Continuous Practices

"Continuous Delivery: Huge Benefits, but Challenges Too" [Chen15a]. This is how Lianping Chen, a software developer of a company called Paddy Power, names his paper. He describes how they (Paddy Power) adopted Continuous Practices, reporting "huge benefits and challenges involved".
The literature about Continuous Practices in general shows that developers often face challenges while adopting CI, CDE or CD. Many of them divided those challenges in different ways depending on their research topic. Some papers, for example, do not involve social challenges, others only focus on architectural challenges. I decided to divide them in 4 different coarse-grained categories, wherein the found ones can be split. Currently, there is little research done on how to better introduce CI,CDE or CD [Chen15a]. As a consequence, "out of the box" solutions are rare.
The following sections are divided in the different kinds of challenges a software development organization may has to face while adopting Continuous Practices Those are organizational, technical, social and architectural challenges. If available, mitigation strategies are provided as well.

5.1 Organizational Challenges

The majority of sources analysed in this seminar paper identifies the breakdown of barriers among the teams as one of the most important organizational challenge. To successfully adopt Continuous Practices, transparency, coordination and a closer collaboration between development teams, operational staff and management has to be established. Team restructuring and the definition of new roles are common mitigation practices [Chen15a]. To establish more transparency, full developer awareness is required, as mentioned in section II.
Having separated development and operation staff causes a very slow failure recovery [ShBZ16]. The introduction of DevOps practices reduces the time between failure discovery and bug fix according to the principle: "You build it, you run it" [ShBZ16]. However, some companies consider the deployment process as a business decision. They believe, only management or financial operators are in charge of the release decision. Asking for permission to release can be a lengthy process, which is obviously contrary to the idea of CD [SBZZ17]. Shahin identifies the lack of experience and skill as another challenge. Using pilot teams, formal training and workshops is a counteracting strategy.

5.2 Technical Challenges

The variety of available tools to implement Continuous Practices is problematic. Especially test tools for user acceptance tests are either non-existent or hardly capable of being integrated into an automated pipeline [SBZZ17]. Besides, manual quality checks are sometimes unavoidable. A fully automated testing infrastructure still is one of the main challenges to achieve full automation [LLIP+16].

As there are no "out of the box" solutions, companies have to design and implement their own CD pipeline. Besides, a "no downtime policy" requires proper hardware, as seamless upgrades are necessary. It is simply not possible, to shut down the system, every time a new version is deployed. This is not just time consuming but connected to a high investment which discourages companies to do CSE [Chen15a]. Shahin et al. reports that security issues in pipeline still remain a big challenge. It has to be noted, that some domains do not allow to truly implement CD: Those are e.g. embedded systems, telecommunication systems, medical and other safety critical systems.

5.3 Social Challenges

The resistance to change or the lack of motivation are obstacles while adopting Continuous Practices [Chen15a] [ClSA15]. To address them, it has to be ensured that team leaders and managers are pushing the new practices and convince their employees of the benefits. Pilot teams and workshops may help to smoothly introduce CSE within the company. Nevertheless, it has been reported that the pressure on developers increases, as they are in charge of having the code always ready for production. Mistrust in the build process and the test results are a common reason for developers to reject CSE [LLIP+16]. Therefore, confidence into the CD process has to be established.
At customer site, demotivated clients make it difficult to introduce Continuous Practices. They need to be involved in the release process, which they primarily consider as a time consuming task [SBZZ17].

5.4 Architectural Challenges

Large, monolithic applications are rarely compatible with Continuous Practices [Chen15a]. Their highly coupled architecture implies cross-functional dependencies which need to be managed in the pipeline. The bigger the application, the higher the complexity that has to be managed within the pipeline. Testing single modules is quite often not possible and monolithic databases become undeployable units. To address these challenges, a redesign is necessary [ShBZ16]. Splitting the application (including the database) into small deployable units, vertical layering or the use of Microservices is necessary. Nevertheless, this is a costly undertaking that often prevents organizations to move software to CD [Chen15b].

6 Conclusion

In this seminar paper, a new software development and deployment approach was presented: Continuous Software Engineering(CSE). CSE enables developers to deliver software more frequently, reduce time-to-market, obtain customer feedback faster, build the right product or to improve the quality of products . In

order to adopt Continuous Practices, (especially Continuous Integration, Continuous Delivery and Continuous Deployment), an approach called "the stairway to heaven" was presented. It is an approach that enables software development organization to make the transition from Traditional Development towards Continuous Deployment. Common practices such as "Full Developer Awareness", "Developer on Call" or "Canary Releases" are incredible helpful to successfully achieve the goal of Continuous Software Engineering.
Also, it became clear that moving an application to CD is challenging. The implementation of CSE requires a vast amount of planning and investment. There is a big number of technical and social challenges, like barriers between teams, a lack of transparency or inadequate tools. Furthermore, CSE requires a shift in culture which is opposed to a general resistance to change mindset. There are neither "out of the box" solutions to implement a continuous deployment pipeline, nor many examples of how organizations exactly adopted Continuous Practices so far. On the other side, there is plenty information about the challenges development teams had to face when they adopted CSE practices.
When studying the available papers about CSE, it became clear that the current definitions are vague. There are no commonly accepted definitions and the boundaries are often not clear. Some authors use CD in a different context than others which sometimes led to confusion.
I do not claim that CDE is the panacea of software development. But there is always a trade off between costs and benefits. Like Agile Software Development, I personally think that CSE is able to revolutionize software engineering and will be widely accepted among developers within the next few years. Nevertheless, further research on how to implement Continuous Practices has to be done. New tools have to be developed to fulfil the requirements of CSE and information about how to move existing applications to CD needs to be gathered.

References

BaWZ15. Len Bass, Ingo Weber und Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional. 2015.

Chen15a. Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2), 2015, S. 50–54.

Chen15b. Lianping Chen. Towards architecting for continuous delivery. In *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*. IEEE, 2015, S. 131–134.

ClSA15. Gerry Gerard Claps, Richard Berntsson Svensson und Aybüke Aurum. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, Band 57, 2015, S. 21–31.

LLIP+16. Eero Laukkanen, Timo OA Lehtinen, Juha Itkonen, Maria Paasivaara und Casper Lassenius. Bottom-up adoption of continuous delivery in a stage-gate managed software organization. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, S. 45.

OlAB12. Helena Holmström Olsson, Hiva Alahyari und Jan Bosch. Climbing the" Stairway to Heaven"–A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*. IEEE, 2012, S. 392–399.

SBZZ17. Mojtaba Shahin, Muhammad Ali Babar, Mansooreh Zahedi und Liming Zhu. Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges. In *Proc. 11th International Symposium on Empirical Software Engineering and Measurement. New York: ACM Press.[To appear]*, 2017.

SCLZ+16. Gerald Schermann, Jürgen Cito, Philipp Leitner, Uwe Zdun und Harald Gall. An empirical study on principles and practices of continuous delivery and deployment. Technischer Bericht, PeerJ Preprints, 2016.

ShBZ16. Mojtaba Shahin, Muhammad Ali Babar und Liming Zhu. The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, S. 44.

ShBZ17. Mojtaba Shahin, Muhammad Ali Babar und Liming Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, Band 5, 2017, S. 3909–3943.

TaKB16. Sajjad Taheritanjani, Stephan Krusche und Bernd Brügge. A Comparison between Commercial and Open Source Reference Implementations for the Rugby Process Model. Technischer Bericht, A University Research Report, 2016.