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 introduces in the early 2000, software development became faster and 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, build the right product or to improve product quality. Therefore, this seminar paper presents the current state of the art concerning Continuous Practices, compares the traditional deployment strategies with the new CSE practices and proposes some tools, that can be used in a CI/CD Pipeline to support CSE. But also, it reveals the barriers an organization has to face while adopting Continuous Practices. CSE not only turns up 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

Deployment Strategies and practices in CI/CD Pipelines 1
Niko Benkler
1 Introduction 3
2 Background 3
2.1 Agile Methodologies 3
2.2 DevOps 4
2.3 Continuous software engineering 4
2.3.1 Continuous Integration 4
2.3.2 Continuous Delivery 5
2.3.3 Continuous Deployment 5
3 From traditional development to Continous Deployment 5

1 Introduction

Today, the software development process has to face many difficult demands. Fast-changing and unpredictable markets, changing customer requirements [ClSA15] and rapidly advancing information technologies [OlAB12] require a faster process of software development. To achieve this, several organizations adopt Continuous Practices in order to extend their agile practices [ClSA15]. 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. As the core of CDE is a deployment pipeline [SCLZ+16], the paper also presents its usual phases and the possible tools, that support each phase. Nevertheless, studying several papers revealed, that CDE not only comes with benefits, but also with huge social and technical challenges [ClSA15]. The paper clarifies them and proposes some mitigation strategies. The remainder of the seminar paper is structured as follows: In section II, we define the terminology. Section III describes a possible transition from traditional deployment to CD based on the 'stairway to heaven' model [OlAB12] and [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 the explanation of a possible pipeline and the available tools which can be used to support the tasks of each phase. Section V discusses the challenges that are caused by CDE and possible mitigation strategies. Finally, I present my conclusion in section VI.

2 Background

Here, I give an overview of the most important keywords. Those keywords are necessary to understand the content of the seminar paper. When studying the given 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 the 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 continuous improvement. Besides, other characteristics such as flexibility, efficiency, speed, the ability to react to fast changing customer requirements and fluctuating market needs [OlAB12] make agile methodologies so attractive for organizations. As mentioned before, agile methodologies were introduced in the

early 2000 [ShBZ17]. So far, many development companies succeeded in implementing agile methodologies. However, agile software development 'only' allows frequent software releases but no continuous releases. The difference becomes clear, when it comes to the comparison of traditional deployment and CD.

2.2 DevOps

"**DevOps** is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal 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 insurance engineers but also IT operations staff [ClSA15] to shorten feedback loop. According to [BaWZ15], the DevOps practices can be differentiated in 5 categories:

- Treat Ops as "first-class citizens". This means, the involvement of operations in the development process.
- Dev has to be more responsible for incident handling 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 continuous deployment.
- Use deployment scripts and other infrastructure code. This code is subjected to the same quality control practices than the application code. As a result, deployment error 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 **(CES)** 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 allows i.a.: to determine new functionality to build, feature prioritization, information about the suitability of the current software architecture, to gather 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 the three development activities: continuous integration, continuous delivery and continuous deployment. Fig. 1 demonstrates the relationship between CI, CDE and CD.

2.3.1 Continuous Integration Continuous Integration **(CI)** is a CSE practice, in which developers integrate and merge their work into a shared repository

very frequently, for example multiple times a day [ShBZ17] [SCLZ+16]. Also, automated builds and test permit to discover integration errors as fast as possible. CI improves the effectiveness of a team as much as the 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 A brief definition of Continuous Delivery **(CDE)** is given in [LLIP+16]: "Continuous Delivery is a software development discipline where you build software in such a way that the 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, such as a less risky release process [LLIP+16] due to automation or the ability to learn from real usage data [OlAB12]. As a result, 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 manually. This is concurrent with the definition of CDE, where it is mentioned that software 'can' be released automatically instead of 'is' released. Therefore, 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), on times a day (nightly build) or several times a week (weekend builds). 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 4. Whereas CD implies the usage of CDE practices, the inversion is not correct [ShBZ17]. Due to barriers such as "lack of automated (user) acceptance tests", "deployment as a business decision", "insufficient level of 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. This topic is discussed in Section 5

3 From traditional development to Continous Deployment

In this section, I will describe a possible transition

References

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

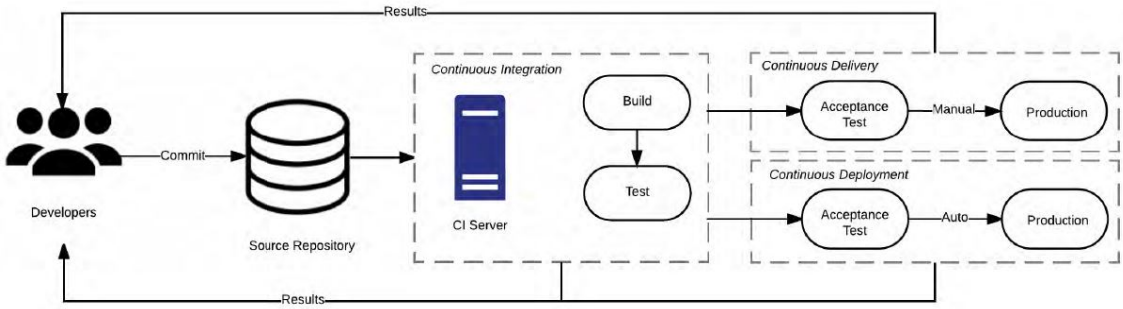

Fig. 1. The relationship between CI, CDE, CD [ShBZ17]

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.

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.