

Seminararbeit

Thema:

Continuous Delivery

Stefan Kruk

geboren am 14.08.1992

Matr.-Nr.: xxxxxxxx

An der Fachhochschule Dortmund im Fachbereich Informatik erstellte

Seminararbeit

im Studiengang Softwaretechnik (Dual)

Betreuer: Dr. Kim Lauenroth

Fachbereich Informatik

Dortmund, 11. Juni 2016

Inhaltsverzeichnis

| | |
|---|------------|
| Abbildungsverzeichnis | ii |
| Glossar | iii |
| 1 Einleitung | 1 |
| 1.1 Grundlagen | 1 |
| 1.2 Problemstellung | 2 |
| 1.3 Ziel der Arbeit | 2 |
| 2 Systematische Literaturrecherche | 4 |
| 2.1 Auswahlkriterien und Suchbegriffe | 4 |
| 2.1.1 Inhaltliche Auswahlkriterien | 4 |
| 2.1.2 Inhaltliche Ausschlusskriterien | 5 |
| 2.1.3 P.I.C.O.C. | 5 |
| 2.1.4 Zusätzliche Anmerkungen zur Recherche | 8 |
| 2.2 Quellen und Suchanfragen | 8 |
| 2.2.1 Suchstring | 8 |
| 2.2.2 Quellen | 9 |
| 2.3 Ergebnisse der Recherche | 10 |
| 3 Continuous Delivery | 12 |
| 3.0.1 Continuous Integration | 12 |
| 3.0.2 Continuous Delivery Pipeline | 14 |
| 3.0.3 Einführung von Continuous Delivery | 14 |
| 4 Zusammenfassung und Ausblick | 17 |
| 4.1 Zusammenfassung | 17 |
| 4.2 Kritische Reflektion | 17 |
| 4.3 Ausblick | 17 |
| Literaturverzeichnis | 18 |
| A Anhang | 19 |
| A.1 Rechercheprotokoll | 19 |

Abbildungsverzeichnis

| | | |
|-----|--|----|
| 3.1 | Continuous Integration | 13 |
| 3.2 | Continuous Delivery Pipeline | 14 |

Glossar

Time-to-Market

Unter dem Begriff Time-to-Market wird die Zeit von der Produktentwicklung bis zur Auslieferung auf dem Markt verstanden. In dieser Zeit müssen Kosten für die Erstellung/Entwicklung aufgebracht werden, es spielt aber keine Umsätze ein. Daher strebt jedes Unternehmen eine möglichst geringe Time-to-Market Zeit an. Insbesondere wenn es um Wettbewerb geht, muss diese Zeit kurz gehalten werden. 2, 16

Kapitel 1

Einleitung

In diesem Kapitel werden zunächst die Grundlagen erläutert, welche für das Verständnis dieser Arbeit notwendig sind. Außerdem werden in den Grundlagen alle wichtigen Begriffe erklärt, die zum Verständnis des Themas beitragen und notwendig sind. Anschließend wird auf die zugrundeliegende Problemstellung eingegangen und darauf aufbauend auf das Ziel der Arbeit.

1.1 Grundlagen

Grundsätzlich ist das in dieser Arbeit behandelnde Thema für jede Person mit einer allgemeinen Informatikausbildung ohne weiteres zu verstehen. Es kann bei dieser Personengruppe, die Kenntnisse über grundsätzlichen Prozess einer Softwareentwicklung vorausgesetzt werden. Trotzdem soll im weiteren Verlauf einige Begriffe genauer erklärt werden.

Delivery

Delivery (zu deutsch Ausliefern) beschreibt das Ausliefern (das Verteilen) von Artefakten. Dabei kann das Artefakt eine ganze Applikation oder nur ein Service in einer Service Orientierten Architektur sein.

Deployment

Unter Deployment (zu deutsch Softwareverteilung) versteht man das installieren, eines Artefaktes. Auch hier kann ein Artefakt eine ganze Applikation oder nur ein

Service sein.

1.2 Problemstellung

Durch fortschreitende Technologien, werden Arbeitsabläufe immer automatisierter. Dadurch wird die Zeitspanne für Time-to-Market immer relevanter und kürzer, wodurch der Wettbewerbsdruck wächst. Daher ist es wichtig eine kurze TTM zu haben.

Eberhard Wolff beschreibt in [6, S. 2 ff.] einen Fall eines fiktiven E-Commerce Unternehmens. Das Unternehmen hatte nur eine große Software, den E-Commerce Shop. Durch neue Angebote und das dauerhaft ändernde Interesse der Kunden mussten neue Funktionen regelmäßig und in möglichst kurzen abständen dem Kunden zugänglich gemacht werden. Dies wurde jedoch durch die Tatsache behindert, dass die Software über die Jahre gewachsen ist und das erneute Ausliefern der Software für eine Funktion sich nicht lohnte. Daher wurde nur einmal im Monat neu Deployed. Der Prozess wurde außerdem dadurch behindert, dass die Qualitätssicherung zwar ein Teil der Softwareentwicklung war, jedoch Tests nur manuell ausgeführt worden sind, wodurch regelmäßig Fehler übersehen wurden.

die Software wurde schließlich mit Fehlern ausgeliefert und es stellte sich erst am nächsten Tag, oder schlimmer nach einer Woche, heraus, dass sie nicht einwandfrei funktionierte. Entwickler mussten also ihre Arbeit unterbrechen und den Fehler finden und beheben. Da jedoch ein wenig Zeit vergangen ist, seit dem die Entwickler an diesem Teil des Codes gearbeitet haben, müssen sie sich erst wieder eingelesen, bis sie den Fehler finden und beheben können.

Das Unternehmen hatte eine große TTM-Zeit und dadurch hohe Kosten. Zusätzlich entstehen fehlerhafte Releases wodurch zusätzliche Kosten bzw. Einbußen entstehen.

1.3 Ziel der Arbeit

In dieser Arbeit soll Continuous Delivery genauer erläutert und dabei folgende Leitfragen beantwortet werden:

1. Was ist Continuous Delivery?
2. Was ist eine Continuous Delivery Pipeline?
3. Wie kann man Continuous Delivery in ein bestehenden Entwicklungsprozess einbinden?

Die erste Leitfrage soll den Begriff Continuous Delivery und seine Herkunft erläutert. Dabei wird kurz auf die Geschichte der Softwareentwicklungsprozesse eingegangen und erläutert wie sich der Prozess zum heutigen unterscheidet. Außerdem wird in diesem Zusammenhang noch einmal erläutert, warum sich die Prozesse verändert haben bzw. verändert werden mussten.

Darauf aufbauend, soll eine Continuous Delivery Pipeline erläutert und aufgebaut werden, anhand dieser wird mit der folgenden und abschließenden Leitfrage, ein bestehender Entwicklungsprozess in Continuous Delivery eingebunden.

Kapitel 2

Systematische Literaturrecherche

Die Systematische Literaturrecherche stellt die Basis der Quellen und Informationen, des in Kapitel 3 **Continuous Delivery** vorgestellten Inhalts dar.

In diesem Kapitel wird daher aufgezeigt, wie die herangezogenen Quellen und Informationen ermittelt, welche Auswahl- und Ausschlusskriterien festgelegt wurden und was für Ergebnisse die entsprechenden Suchanfragen gebracht haben.

2.1 Auswahlkriterien und Suchbegriffe

Um die Auswahl der Literaturen zu Filtern, wird zunächst allgemeine Auswahl- und Ausschlusskriterien definiert, mit denen die im **Rechercheprotokoll** angegebenen Suchergebnisse begründet werden. Anschließend wird der Zugrundlegende Ansatz (P.I.C.O.C.) genauer erläutert und darauf aufbauend Suchbegriffe in Deutsch und Englisch definiert.

2.1.1 Inhaltliche Auswahlkriterien

Folgende Inhaltliche Auswahlkriterien wurden für die Recherche festgelegt: spacing

P1 Dokument ist über oder hat direkten Bezug zu Continuous Delivery

P2 Dokument beschreibt die Einsatzmöglichkeiten von Continuous Delivery

P3 Dokument beschreibt wichtige Technologien für den Einsatz von Continuous Delivery

Mit Hilfe der Auswahlkriterien wird im **Rechercheprotokoll** die Relevanz der gefundenen Materialien begründet. Sie werden über die Buchstaben a) bis c) referenziert.

2.1.2 Inhaltliche Ausschlusskriterien

Folgende Inhaltliche Ausschlusskriterien wurden für die Recherche festgelegt. spacing

- N1** Dokument ist zu allgemein und hat nur am Rande etwas mit dem Thema zu tun (Bsp.: Enthält den Begriff nur in Referenzen)
- N2** Dokument beschreibt wie ein Werkzeug und/oder Framework aufgebaut ist und funktioniert.
- N3** Inhaltsangabe, Einleitung, Fazit oder Abstract sind nicht Aussagekräftig bzw. lassen keine Hinweise auf den Einsatz oder der Beschreibung von Continuous Delivery zu
- N4** Inhalt trägt nicht zur Beantwortung der Leitfragen bei. (Bsp.: Das Dokument ist ein Erfahrungsbericht, enthält jedoch keine konkreten Erläuterungen oder Verweise auf Dokumente, die den Prozess genauer beschreiben.)
- N5** Dokument ist nicht in Deutsch oder Englisch (Material kann aufgrund der Sprachbarriere nicht verwendet werden)

Mit Hilfe der Ausschlusskriterien wird im **Rechercheprotokoll** die Irrelevanz der gefundenen Materialien begründet. Sie werden über die Buchstaben d) bis j) referenziert.

2.1.3 P.I.C.O.C.

Die Suchbegriffe werden mit Hilfe des PICOC-Ansatzes (siehe [4]) ermittelt.

Population

Die Population, zu Deutsch etwa "Bevölkerung", beschreibt eine Teilmenge von relevanten Personen, wie Tester, Manager, Novizen oder Experten. Aber auch Applikationsfelder wie IT-Systeme, Command und Control Systeme oder Industrielle

Gruppen wie Telekommunikations- oder kleine IT-Unternehmen.

Bezüglich der Population werden hier die sogenannten "SSStackholder" weggelassen. Da das Thema jedoch stark mit dem Thema DevOp-Teams tangiert, wird hier zusätzlich zu diesem Thema Suchanfragen gestellt. Eine Einschränkung des Themas auf DevOps besteht jedoch nicht.

Intervention

Bei der Intervention handelt es sich um die eingesetzten Methoden, Werkzeugen, Technologien oder Prozeduren für ein bestimmtes Problem. Im Rahmen von Continuous Delivery bedeutet dies, die Werkzeuge die nötig sind um eine Continuous Delivery Pipeline aufbauen und durchführen zu können. Grundsätzlich sollen Werkzeuge erläutert werden, welche zur Verwaltung, Bauen, Testen und Ausliefern dienen.

Comparison / Vergleich

Der Vergleich (Comparison) beschreibt die Werkzeuge einer Kontrollgruppe, welche mit denen aus der Intervention verglichen werden. Konkret werden die Gruppen aufgrund der TTM verglichen. Anzumerken sei, dass die Vergleichsgruppe ohne einen automatisierten Prozess Softwareprojekte durchführt.

Outcomes / Auswirkung

Bei der Auswirkung soll mit Hilfe von Zahlen und Faktoren der Vergleich erläutert werden zwischen der Intervention und der Kontrollgruppe. Hier kann zum Beispiel die Zeitspanne des Time-to-Market verglichen werden, was wiederum einen Einblick in die Kosten für die Implementierung/Auslieferung eines Produktes/einer Funktion gibt.

Context / Kontext

Der Kontext ist hier die Software-Entwicklung in Bezug auf Aufwand und Kosten der Produktion. Bei Continuous Delivery spielt ebenfalls der Aufbau des Teams

eine Rolle. Wie bei der **Population** spielen hier DevOp-Teams eine zentrale Rolle. Zusätzlich sei hier auf **Intervention** und die dortigen Werkzeuge verwiesen.

Suchbegriffe

Im Nachfolgenden sind für jeden, der in Abschnitt 2.1.3 P.I.C.O.C genannten Aspekte Synonyme in Deutsch und Englisch festgehalten, welche die Basis für die verwendeten Suchanfragen bilden.

| Kategorie | Deutsche Begriffe | Englische Begriffe |
|--------------|--|---|
| Population | DevOps | |
| Intervention | <ul style="list-style-type: none"> • Continuous Delivery • Docker • Jenkins • Deployment • pipeline | |
| Comparison | <ul style="list-style-type: none"> • Manuel (Deployment) | |
| Outcomes | <ul style="list-style-type: none"> • Kosten • Tests • Zeit | <ul style="list-style-type: none"> • costs • tests • time / duration |
| Context | <ul style="list-style-type: none"> • Technik • Prinzipien • Praxis • Anwendung | <ul style="list-style-type: none"> • techniques • principles • practice • usage |

Der

Begriff Deployment ist sowohl in Intervention und Comparison, da dieser ein zentraler Begriff in beiden Mengen ist und unterschiedlich verwendet werden kann.

2.1.4 Zusätzliche Anmerkungen zur Recherche

Für diese Arbeit wurden zusätzlich folgenden Einschränkungen, für die Durchführung der systematischen Suche festgelegt:

Zugänglichkeit Materialien müssen entweder öffentlich oder für den Personenkreis, für die diese Arbeit angefertigt wird, ohne weitere Einschränkungen, wie ein notwendiges Login, zugänglich sein. da ansonsten der Beschaffungsaufwand zu hoch ist und die Materialien nicht für andere Studenten bzw. den Dozenten zugänglich wären.

Auswahl der Materialien Materialien werden anhand der Inhaltsübersicht, Einleitung, Fazit oder eines Abstracts ausgewählt, da ein einlesen in einzelne Kapitel zu viel Zeit beanspruchen würde.

Suchergebnisse Bei auffinden von Großen Mengen bei der Suche, wird zunächst versucht durch eventuelle Filtermöglichkeiten, die Relevanz der Materialien zu ordnen und die ersten 20 Resultate begutachtet. Dabei wird die Qualität der Ordnung und die Effizienz des zugrunde liegenden Algorithmus der Suchmaschine überlassen. Sollten keine Filtermöglichkeiten vorhanden sein, wird anhand der Kurzbeschreibungen und der Titel die ersten 20 besten Treffer ausgewählt.

2.2 Quellen und Suchanfragen

2.2.1 Suchstring

In Abschnitt 2.1.3 Suchbegriffe wurden Suchbegriffe festgelegt, auf denen die Literaturrecherche basiert. Diese werden zu nächst mit einem “ODER” bzw. “OR,, verknüpft. Im Zweiten Schritt werden die Suchbegriffe mit einem “UND” bzw. “ÄND,, verknüpft.

Da bestimmte Begriffe mit verschiedenen Kontexte in Verbindung gebracht werden können. Wird zunächst ein Suchstring aufgebaut, der als Erstes Element (“Continuous Delivery,, OR “Deployment,,) besitzt. Nachfolgend werden nun die einzelnen

Suchstrings aufgelistet, die verwendet wurden, um die systematische Literaturrecherche durchzuführen. Da Suchmaschinen einen komplexen Algorithmus aufweisen, wird ebenfalls davon ausgegangen, dass auch eine Aneinanderreihung von den in 2.2.1 Suchstring Ausdrücken (immer mit dem Führenden "Continuous Delivery,, Oder "Deployment") zum gewünschten Ergebnis führt. Dies beruht auf den Eigenschaften moderner Suchalgorithmen. Nach einer ersten Suche, hat sich ergeben, dass einige Begriffe irreführend für die Suchmaschinen sind, wodurch Materialien gefunden wurden, welche absolut nichts mit dem Thema zu tun haben. Daher sind nur die folgenden Suchanfragen von Bedeutung.

$$S_{Komplex} = ("Continuous Delivery" \text{ OR } "Deployment")$$

AND

("SSH,, OR "FTP,, OR "Deployment,, OR "Kosten,, OR "costs,, OR "Tests,, OR "Zeit,, OR "time,, OR "duration,, OR "Fehler,, OR "error,, OR "Qualitätssicherung,, OR "quality assurance,, OR "Technik,, OR "techniques,, OR "Prinzipien,, OR "principles,, OR "Praxis,, OR "practice,, OR "Anwendung,, OR "usage")

$$S_{deployment} = ("Continuous Delivery" \text{ OR } "Deployment")$$

AND

("Deployment,,)

$$S_{pipeline} = ("Continuous Delivery" \text{ OR } "Deployment")$$

AND

("Pipeline,,)

2.2.2 Quellen

In [4] wurden einige elektronische Standardquellen der Informatik genannt. In der nachfolgenden Tabelle, werden diese noch einmal aufgelistet und kurz begründet, warum jeweilige Quellen gewählt bzw. nicht gewählt wurde.

| Quelle | gewählt | Begründung |
|---|---------|---|
| ACM Digital library | ✗ | Nicht frei zugänglich |
| Citeseer library (citiseer.ist.psu.edu) | ✓ | Fokus auf Informatik, PDFs kostenlos verfügbar |
| EI Compendex (www.engineeringvillage2.org) | ✗ | Registrierung erforderlich |
| IEEEExplore | ✓ | über die FH frei zugänglich, relevante Quelle für Informatiker |
| Inspec (www.iee.org/Publish/INSPEC/) | ✗ | Registrierung erforderlich |
| Google scholar (scholar.google.com) | ✓ | Großes Angebot, <u>meist</u> als PDF frei zugänglich |
| ScienceDirect (www.sciencedirect.com) | ✗ | PDFs sind nur teilweise frei zugänglich. |

2.3 Ergebnisse der Recherche

In der Nachfolgenden Tabelle sind, nach Suchmaschine geordnet, die Ergebnisse der systematischen Literaturrecherche. Es wird der Suchstring, die Anzahl der gefundenen Ergebnisse, die Anzahl der betrachteten Ergebnisse, die Anzahl der relevanten Ergebnisse, die Anzahl der gewählten Ergebnisse und das Datum an dem die Suche durchgeführt worden ist.

| Citeseer library | | | | | |
|------------------|---------|------------|----------|---------|------------|
| Suchstring | gesamt | betrachtet | relevant | gewählt | Datum |
| $S_{komplex}$ | 350.724 | 0 | 0 | 0 | 13.05.2016 |
| $S_{deployment}$ | 207.039 | 0 | 0 | 0 | 13.05.2016 |
| $S_{pipeline}$ | 120.126 | 0 | 0 | 0 | 13.05.2016 |

| IEEEExplore | | | | | |
|------------------|--------|------------|----------|---------|------------|
| Suchstring | gesamt | betrachtet | relevant | gewählt | Datum |
| $S_{komplex}$ | 0 | 0 | 0 | 0 | 13.05.2016 |
| $S_{deployment}$ | 60 | 10 | 10 | 4 | 13.05.2016 |
| $S_{pipeline}$ | 22 | 3 | 3 | 2 | 13.05.2016 |

| Google scholar | | | | | |
|------------------|--------|------------|----------|---------|------------|
| Suchstring | gesamt | betrachtet | relevant | gewählt | Datum |
| $S_{komplex}$ | 1.580 | 1 | 0 | 0 | 13.05.2016 |
| $S_{deployment}$ | 224.00 | 2 | 1 | 1 | 13.05.2016 |
| $S_{pipeline}$ | 80.800 | 2 | 1 | 1 | 13.05.2016 |

Leider erga-

ben die suchen, mit den oben angegebenen Suchstrings, keine eindeutigen Ergebnisse. Eine Konkrete suche nach einzelnen Begriffen führte zu einer noch größeren Breite an Informationen, welche auf Grund der in [2.1.2 Inhaltliche Ausschlusskriterien](#) erläuterten Kriterien ausgeschlossen wurden und daher nicht dokumentiert wurden.

Kapitel 3

Continuous Delivery

Martin Fowler schreibt [2]:

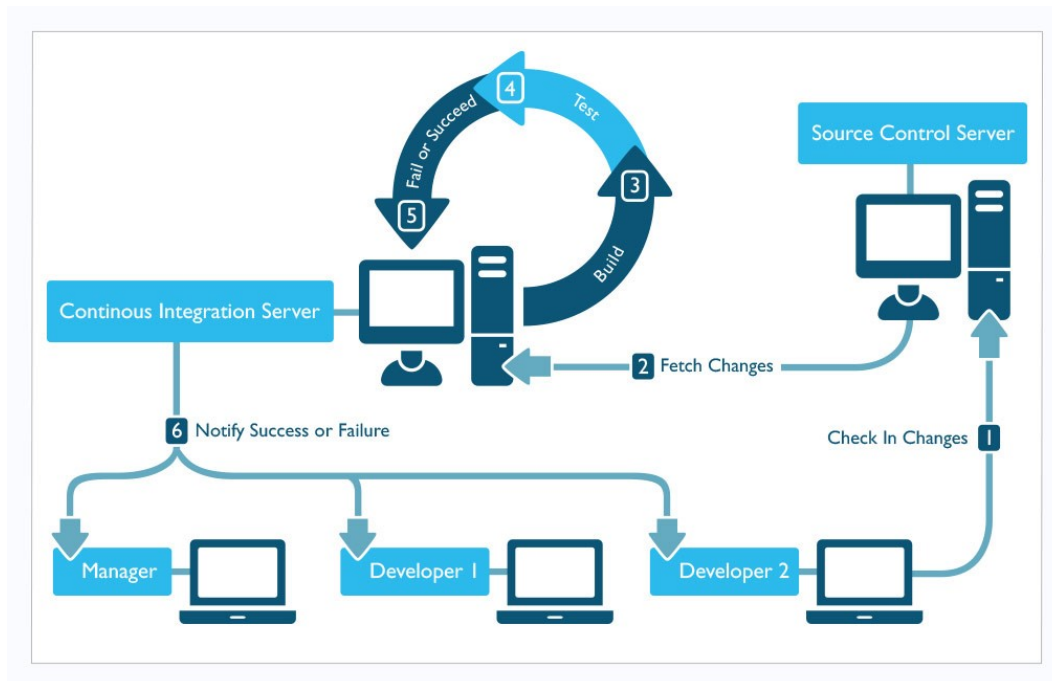
„Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time. [...] You achieve continuous delivery by continuously integrating the software done by the development team, building executables, and running automated tests on those executables to detect problems. Furthermore you push the executables into increasingly production-like environments to ensure the software will work in production. To do this you use a Deployment Pipeline.“

Wie von Martin Fowler zu lesen ist, ist Continuous Integration ein Teilprozess von Continuous Delivery und beschreibt dessen Grundlagen. Bevor Continuous Delivery genauer erläutert werden kann, muss zunächst der Begriff Continuous Integration erläutert werden.

3.0.1 Continuous Integration

„At all times you know where you are, what works, what doesn't, the outstanding bugs you have in your system “[1].

Das ist die Aufgabe von Continuous Integration. Um dies zu erreichen ist es notwendig jede Teilaufgabe innerhalb der Entwicklung in den Prozess mit einzubinden. Das Ziel ist es so früh wie möglich Bugs zu erkennen und diese zu beheben. Dafür muss jeder Prozess automatisiert werden. Vor allem muss es möglich sein, Tests automatisiert durchführen zu können. Eine einfache Continuous Integration zeigt folgendes Bild:



Quelle: <https://insights.sei.cmu.edu/devops/2015/01/continuous-integration-in-devops-1.html>

Abbildung 3.1: Continuous Integration

Wie zu erkennen ist, sorgt ein zentraler „Continuous Integration Server“ für das Bauen und Testen der Software und informiert die zuständigen Entwickler über den Status. In der Regel wird dies bei jeder Code Änderung durchgeführt, sodass direkt erkannt wird, ob eine Änderung des Codes zu einem Erfolg oder einem Fehlschlag führt.

Continuous Delivery ist eine natürliche Erweiterung von Continuous Integration. Trotzdem unterscheiden sich die beiden Begriffe nicht wirklich von einander. Während bei der Continuous Integration wert darauf gelegt wird, Software möglichst Fehlerfrei zu erzeugen, wird bei Continuous Delivery darauf wert gelegt, Software möglichst regelmäßig zu deployen. Continuous Delivery beinhaltet Continuous Integration und erweitert diese um das ausliefern. In der nachfolgenden Sektion, wird noch einmal auf den unterschied zwischen Continuous Integration und Delivery eingegangen

3.0.2 Continuous Delivery Pipeline

Die Continuous Delivery Pipeline beschreibt die Teilprozesse, welche durchlaufen werden müssen, um Continuous Delivery durchführen zu können. Dabei wird die Pipeline automatisch durchlaufen. Die Folgende Abbildung zeigt eine mögliche Pipeline. Das Deployen in die Produktionsumgebungen „PRODBLUE“ und „PROD-GREEN“ muss jedoch manuell, durch klicken auf Trigger, erfolgen.



Quelle: <https://blog.codecentric.de/en/2012/04/continuous-delivery-in-the-cloud-part1-overview/>

Abbildung 3.2: Continuous Delivery Pipeline

Wie man in der Abbildung sehen kann beinhaltet die Pipeline alle nötigen Schritte, welche zuvor in **Continuous Integration** besprochen wurden. Hier sei noch einmal erwähnt, dass Continuous Integration alle Prozesse, bis auf den letzten (das Deployment), beinhaltet. Continuous Integration und Delivery unterscheiden sich, wie schon zu vor erwähnt, nur beim letzten Schritt, dem Deployen. Dieser wird jedoch bei Continuous Delivery Manuell ausgeführt. Unter ausgeführt ist hierbei zu verstehen, dass ein Prozess, hier durch einen Klick, gestartet wird, welcher die Software, bzw. das Artefakt, automatisch in die Produktion bringt.

3.0.3 Einführung von Continuous Delivery

In Kapitel 1.2 **Problemstellung** wurde bereits ein Fall beschrieben, in der kein Continuous Delivery eingesetzt wird. Darauf aufbauend wird nun Schritt für Schritt die vorhandene Softwareentwicklung in Continuous Delivery überführt.

Damit das Dependencie Management und automatisieren von Tests bzw. das ausführen zusätzlicher Aktionen erleichtert wird, wird zunächst ein Build-Management-Tool eingeführt. Dadurch wird sichergestellt, dass Software idempotent gebaut werden kann. Der Build-Prozess ist dadurch Standardisiert und kann beliebig oft wiederholt werden.

Wie bereits erläutert wurde, ist in diesem Beispiel die Qualitätssicherung zwar ein Teil der Softwareentwicklung, jegliche Tests werden jedoch nur manuell ausgeführt, was zu regelmäßigen, unentdeckten Fehlern führt. Daher muss zunächst einmal dafür gesorgt werden, dass Unit-Tests geschrieben werden, welche vor dem Bauen der Software, die Code Qualität und Richtigkeit überprüft. Test-Driven-Development (TDD)¹ ist einer der Möglichkeiten, sicherzustellen, dass Tests regelmäßig geschrieben werden. Zusammen mit dem eingeführten Build-Management-Tool können nun, vor dem Bauen der Software, die Unit Tests durchlaufen und dadurch der Code der Software überprüft werden.

Nun ist zu mindestens Sichergestellt, dass diese Art der Tests automatisiert ablaufen und nicht mehr manuell durchgeführt werden müssen. Jedoch müssen noch Akzeptanz-, Performanz- und Integrations-Tests automatisiert werden. Diese Tests können jedoch nicht immer durch das Build-Management-Tool abgedeckt werden. Daher Wird nun ein „Build and Management“ Werkzeug wie Jenkins² eingesetzt wird, welches sowohl die Software baut, unter anderem mit dem eingeführten Build-Management-Tool, als auch die anderen Tests abbilden kann. Wie die Software genau funktioniert, soll hier jedoch nicht weiter erläutert werden. Es sei nur erwähnt, dass es durch Konfiguration und Plugins möglich ist, die oben genannten Tests innerhalb von Jenkins abzubilden. Es wurde eine *Continuous Delivery Pipeline* erschaffen. Die Abbildung 3.2 Continuous Delivery Pipeline zeigt ein ausschnitt aus dem „Build and Management“ Werkzeug Jenkins. Außerdem ist es durch Jenkins möglich jeden Schritt zu Überwachen und zu Protokollieren, sowie nach jedem

¹TDD wird hier nur erwähnt und nicht weiter erläutert. Es sei auf Fachliteratur zu diesem Thema verwiesen.

²siehe: <https://jenkins.io/>

Schritt zu stoppen, sollte ein Fehler auftreten. Dadurch ist es möglich frühzeitig Fehler zu erkennen und diese zu beheben.

Es wurde nun dafür gesorgt, dass sowohl das Bauen, als auch jegliche Tests automatisiert wurde. Dadurch kann die Software Standardisiert gebaut und getestet werden. Durch die Automatisierung ist es zusätzlich möglich die Time-to-Market Zeitspanne deutlich zu kürzen, da automatisch durchgeführten Tests, meistens kürzer und genauer sind, als wenn man diese manuell ausgeführt hätte. Außerdem werden unter anderem dadurch Fehler frühzeitig erkannt und können behoben werden, bevor die Software in Produktion geht. Durch die durchgeführten Änderungen am Entwicklungsprozess, ist es nun möglich bei jeder Änderung des Quellcodes, die Software standardisiert zu bauen und zu Testen. Dadurch erhält der, bzw. die Entwickler regelmäßig und in kurzen Abständen eine Rückmeldung, ob die durchgeführten Änderungen am Code zu Problemen führen oder die Software weiterhin funktioniert. Dies führt dazu, dass regelmäßig releasefähige Software erzeugt wird.

Kapitel 4

Zusammenfassung und Ausblick

4.1 Zusammenfassung

4.2 Kritische Reflektion

4.3 Ausblick

Literaturverzeichnis

- [1] FOWLER, Martin: *Continuous Integration*. 01 May 2006. – visited: 08.06.2016
- [2] FOWLER, Martin: *ContinuousDelivery*. 30 May 2013. – visited: 08.06.2016
- [3] FOWLER, Martin: *DeploymentPipeline*. 30 May 2013. – visited: 08.06.2016
- [4] KITCHENHAM: *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007
- [5] WIESMANN, Prof. Dr. D.: *Skript der Veranstaltung SWT D*. FH-Dortmund. 2014
- [6] WOLFF, Eberhard: *Continuous Delivery - Der pragmatische Einstieg*. 1. Auflage. dpunkt.verlag, 2015. – ISBN 978-3-86490-208-6

Anhang A

Anhang

A.1 Rechercheprotokoll

Kriterien der Kategorie **P** sind **Inhaltliche Auswahlkriterien**.

Kriterien der Kategorie **N** sind **Inhaltliche Ausschlusskriterien**. Doppelte Einträge wurden Grau hinterlegt.

| IEEEExplore | | | |
|------------------|--|--------------------|-----------|
| Suchstring | Titel | Author | Kriterium |
| $S_{deployment}$ | End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery | Miteshi Soni | P2/P3 |
| $S_{deployment}$ | Continuously delivering your network | Steffen Gebert | P2 |
| $S_{deployment}$ | Continuous Delivery: Huge Benefits, but Challenges Too | Lianping Chen | P2 |
| $S_{deployment}$ | Towards Architecting for Continuous Delivery | Lianping Chen | P |
| $S_{pipeline}$ | Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery | Valentina Armenise | N2 |
| $S_{pipeline}$ | Continuous Delivery: Huge Benefits, but Challenges Too | Valentina Armenise | P2 |

| Google Scholar | | | |
|------------------|---|--------------------|-----------|
| Suchstring | Titel | Author | Kriterium |
| $S_{deployment}$ | Continuous Integration | Martin Fowler | P1 |
| $S_{pipeline}$ | Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery | Valentina Armenise | P3 |
| $S_{pipeline}$ | (IEEEExplore) Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy) | Steve Neely | P2 |