h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Informatik–

Anforderungen und Eigenschaften einer qualitativen CI/CD Pipeline

Begleitseminar "Problemlösung und Diskussion"

vorgelegt von

Manuel Plonski

Matrikelnummer: 756989

Referent : Prof. Dr. Ralf Hahn

ERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 28. November 2019

Manuel Plonski

INHALTSVERZEICHNIS

Seitenbeschriftun und Seitenzahlen stimmen noch nicht ganz

Teil I

SEMINARARBEIT

1 EINLEITUNG

Das Thema der Digitalisierung ist heute nicht mehr wegzudenken, wenn es in den Unternehmen darum geht in der wachsenden Dynamik des Marktes passend zu agieren. Dabei muss auch die Softwareentwicklung sich diesem Trend anpassen und so von zeit- und kostenintensiven Softwareprojekten zu einer agilen Entwicklung mit vollautomatisierten Prozessen umschwenken. Der moderne Softwareentwicklungsprozess wird immer weiter automatisiert. Hier sind "Continuous Integration" und "Continuous Deployment/-Delivery" (CI/CD) schon weit verbreitet. Unter anderem ermöglichen Konzepte wie CI/CD schnellere Entwicklungsgeschwindigkeiten und erhöhen so die Chance, Fehler frühzeitig zu erkennen.[3]

MOTIVATION

Wie bei allen neuen Methoden mit den dazugehörigen Werkzeugen muss die korrekte Anwendung erlernt und beachtet werden. Denn die Vorteile kommen nur dann zum Tragen, wenn diese eine konsequente und richtige Anwendung finden. Wird das Konzept von CI/CD nicht durchdrungen oder inkonsequent umgesetzt, so können verschiedene Probleme auftreten, welche dann zu Lasten von Zeit, Projektbudget und Effizienz gehen. Beispielsweise kann fehlendes Wissen über die Funktionsweise des CI/CD Systems im Falle einer Störung zum Stillstand des Entwicklungsprozesses führen.

ZIEL DER ARBEIT

Grundsätzlich stellt sich die Frage, wie die Konzeption und das Design eines CI/CD Systems aussehen sollte, um die Vorteile der Methode bestmöglich zur Geltung zu bringen und sich Probleme in der Anwendung von Verfahren und Werkzeugen vermeiden zu lassen. Daraus ergibt sich die Fragestellung nach den Qualitätseigenschaften eines CI/CD System welche ich ins Zentrum meiner Seminararbeit stellen möchte. Hieraus ergeben sich unter anderem folgende Fragen:

- Was bedeutet Qualität für ein CI/CD System?
- Welche Anforderungen stellen sich an eine CI/CD Pipeline mit hoher Qualität?
- Lassen sich Basisanforderungen aufstellen, unabhängig von dem Einsatzgebiet der Pipeline?

Ziel ist die Ausarbeitung von Anforderungen und Qualitätsmerkmalen an ein CI/CD System, welche die Entwicklungsgeschwindigkeit steigert, ohne die Softwarequalität zu kompromittieren. Anforderungen an eine Qualitätsvolle CI/CD Pipeline aufstellen.

VORGEHEN

In einem ersten Schritt betrachte ich den Softwareentwicklungsprozess und die Automatisierungsmöglichkeiten im Allgemeinen. Anschließend werden die Vorteile und Probleme unter dem Einsatz von CI/CD erarbeitet, um dann entsprechende Qualitätsmerkmale abzuleiten und Anforderungen an die Tools, Standards und Methoden eines CI/CD Systems zu formulieren.

2

ANFORDERUNGEN UND EIGENSCHAFT AN DIE ERGEBNISSE

Um Qualitätsmerkmale und Anforderungen für eine Vielzahl verschiedener Anwendungsszenarien für CI/CD umsetzbar zu machen, ist es notwendig, sie zunächst sehr universell zu verfassen.

ANFORDERUNGEN

- Eine Anforderung darf nicht spezifisch auf eine Programmiersprache (beispielsweise Java, Go, C++) oder eine Systemumgebung ausgerichtet sein.
- Ebenso sollte sie unabhängig von Prozessen, eingesetzten Tools und der Systemarchitektur formuliert werden.
- Die Qualitätsanforderungen soll dabei so erarbeitet werden, dass eine Umsetzung nicht nur in komplexen Umgebungen wie einem Datencenter möglich ist, sondern auch auf einem einfachen Entwicklungsrechner einen Betrieb ermöglichen kann.
- Auch der Kosten/Nutzenaspekt sollte in der Leistungsanforderung einer CI/CD Systems berücksichtigt werden.
- Die Anforderung und Qualität soll nicht an einzelnen Komponenten des CI/CD Systems ausgerichtet werden, sondern an der Funktionsweise des Gesamtsystems. So werden gegebenenfalls positive wie negative Eigenschaften von Tools, wenn sie für sich allein betrachtet werden, ausgeblendet und nur die Gesamtleistung der Pipeline bewertet.
- Auch ist die “zeitlose” Anwendbarkeit dieser Anforderung eine wichtige Eigenschaft: Die Anforderungen sollte unabhängig von der rasanten Entwicklung auf dem Toolmarkt Bestand haben.

3

AUTOMATISIERUNG IM SOFTWARE-ENTWICKLUNGSZYKLUS

In der Softwareentwicklung gibt es eine Vielzahl von Modellen zur Unterstützung des Softwareentwicklungsprozess. Wie im ersten Kapitel erwähnt, sollen die Anforderungen so generisch wie möglich formuliert werden. Um eine größtmögliche Vielzahl verschiedener Modelle abzudecken, abstrahiere ich deren Komponenten aus dem Modell und versuche ihre Funktion möglichst unabhängig von der Methode darzustellen.

DARSTELLUNG DES SOFTWARE-ENTWICKLUNGSPROZESSES

Bei der Herleitung dieser generischen Sichtweise auf den Software-Entwicklungsprozess habe auf Elemente des Wasserfallmodell aus der Publikation von Royce, Winston W: *"Managing the development of large software systems: concepts and techniques"*[5] zurückgegriffen. Auch wenn er selbst dieses Modell als **"riskant und fehlerbehaftet"** beschreibt und heute andere Methoden sich etabliert haben. Dabei konzentriere ich mich nur auf die Phasen dieses Modells, unabhängig von der Anordnung, Iteration, ihrem Kontext oder Umfang und auch der Philosophie der jeweiligen Methode. Diese einzelnen Schritte lassen sich in ähnlicher Form in vielen anderen Methoden wiederfinden.

Projektphasen

Die Projektphasen des Wasserfallmodells werden in Anforderungsanalyse, Systemdesign, Programmierung, Testing und Betrieb unterschieden.

ANFORDERUNGSANALYSE In diese Phase geht es darum, Anforderungen zu sammeln und zu analysieren. Dies geschieht in Form von Texten oder Modellen, die der Strukturierung und Klassifizierung dienen.

SYSTEMDESIGN Hier wird die Architektur der Module, Schnittstellen und Daten festgelegt, die der Spezifikation aus der Anforderungsanalyse genügen sollen.

IMPLEMENTIERUNG Die Programmierung mit dem dazugehörige Modultest sind Bestandteil der Entwicklungsphase der Implementierung.

SOFTWARETEST In der Testphase wird ein System oder eine Komponente gegen die zuvor spezifizierten Anforderungen überprüft. Das Testergebnis dient der Behebung von Softwarefehlern um ein fehlerfreies System in Produktion zu übernehmen.

BETRIEB Der Betrieb um fasst die Auslieferung und Wartung, Betrieb der Software.

AUTOMATISIERUNG IM ENTWICKLUNGSPROZESS

Es gibt verschiedene weit verbreitete Automatisierungspraktiken. Dazu gehören[2]:

- CI (*Continuous Integration*)
- CDE (*Continuous Delivery*)
- CD (*Continuous Deployment*)

Die Idee bei *Continuous Integration* besteht darin, wiederholt (mehrmals täglich) die Software in einer kontrollierten definierten Umgebung automatisiert zu integrieren und zu testen (automated build). Die Tests können den Modul- Integration- und auch den Abnahmetest umfassen. Durch diese Verfahren können Fehler durch Codeänderungen automatisch erkannt werden, die sonst durch aufwendige, manuelle Testverfahren oder sogar erst im Betrieb erkannt worden wären. Je nach Umfang kann CI das ganze Volumen von Testszenarien abwickeln und so die Testphase voll automatisiert abdecken.

Bei dem *Continuous Delivery* geht es darum, eine neu entwickelte Softwareversion zur Auslieferung bereitzustellen. Die Installation und der Betrieb der Software erfolgt jedoch manuell.

Continuous Deployment erweitert das *Continuous Delivery* um die automatische Installation und Konfiguration auf dem Zielsystem. Mit *Continuous Deployment* werden Komponenten aus der Phase „Betrieb“ automatisiert und das Fehlerrisiko bei der Installation verringert.

Diese Grundprinzipien können erweitert werden: so lassen sich z.B. mit Hilfe von statischer Codeanalyse automatisiert Fehler oder Sicherheitsprobleme erkennen (CI). Ein Beispiel für die Erweiterung der Phase „Betrieb“ ist das automatisierte Monitoring von Produktionssystemen, welches z.B. Alarmmeldungen in Fehler- und Problemsituationen erzeugt.

In der Implementierungsphase gibt es ebenfalls Möglichkeiten zur Automatisierung. Sind die Ergebnisse des Designs in einer geeigneten Form dokumentiert worden (z.B. UML oder Open API), lassen sich Teile des Codes wie Datenmodelle oder Server – Boilercode automatisch generieren.

Für die Phase der Anforderungsanalyse und des Systemdesigns gibt es ebenfalls eine Vielzahl von Tools, welche der Ergebnisdokumentation dienen – sie können die Aktivitäten zur Analyse und Design aber nur unterstützen und nicht automatisieren.

Teil II

WORK IN PROGRESS

4

VORTEILE/NACHTEILE VON AUTOMATISIERUNG IM ENTWICKLUNGSZYKLUS

Dieses Kapitel ist momentan nur eine Ansammlung von einzelnen Vorteilen und Problemen die durch CI/CD auftreten können. Die Ansammlung ist noch nicht vollendet, jedoch kann man schon erahnen in welche Richtung die Qualitätsmaßnahmen sich entwickeln. . . . Mit der Rechtschreibung in diesem Kapitel hab ich es auch nicht so ernst genommen.

VORTEILE

The 2019 Accelerate State of DevOps report, for which Dora surveyed 3,000 developers, found that, comparing the elite group to poor performers, the former achieved 208 times more frequent code deployments, 106 times faster lead time from committing code to deployment and 2,604 times faster to recover from incidents. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD

With CI/CD you get cloud-native things like optimisation, scalability and elasticity, and performance. CI/CD ermöglicht auto cloud deployment. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD

Documentation with Markdown is wonderful.: The source is stored alongside the code, and all the DevOps best practices apply. Developers are more invested in documentation because they know what to do with a pull request (versus reviewing and editing in some alien documentation tool). Finally, this encourages teams to write docs at the same time as the source code. https://www.cloudbees.com/blog/silos-begone-road-devops-autodesk-and-lessons-learned-along-way

NACHTEILE

Lack of experience and skill: Source: Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices

Unexpected Deployment Ausversehen auf master gepushed. Source: Manuel + A4

Infrastructure Make sure you invest in the underlying infrastructure, the correct tooling. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD + Manuel

Storage: Durch Automation fallen viele Artefakte an. Werden diese nicht gemanaged wird auch viel Speicher anfallen. Source: A4

Big Red Button: Es sollte einen Stop knopf geben. Source: Manuel + A4

DB Migration: Schema Migration können beim deployment in ein envirgoment auftreten. Race condition bei vielen daten. https://dzone.com/articles/a-year-of-continuous-deployment-lessons-learned

POOR Test Coverare TEST: Durch Falsche Test Coverage Tests die letztendlich nur Smoketests waren, sah es so aus als ob die Tests gut gelaufen sind. https://dzone.com/articles/a-year-of-continuous-deployment-lessons-learned

Missing Documentation “The mistake a lot of people make is they think it is all about building quickly and delivering fast, but they forget to document properly, so nobody knows what they have actually done,” she says. “It takes time at the beginning to automate things and write scripts. But once you’ve got that right, you can repeat it as many times as you want,” she says. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD

Speed(Mental) “Teams that go too fast before they are ready just break things faster, and they don’t get fixed. It’s definitely about going as fast as you can, but no faster. That’s the biggest lesson I’ve learned,” he says. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD

Dependencymanagement was hard. Their siloed history meant teams had their own copies of shared binaries, and in some cases had modified the source locally. Finding owners for shared components, and unifying the source code and binaries back into a single GH repo and Artifactory folder continues to be a challenge. https://www.cloudbees.com/blog/silos-begone-road-devops-autodesk-and-lessons-learned-along-way

Perspective Shift Schwer die Leute einzuarbeiten. Lieber dinge benutzen mit denen schon gearbeitet wird. Source: Awesome Paper[6]

BIG BANG trotz CI Haltet die env so gleich wie in Prod. Source: Präsi ... muss link noch finden.

Desaster Recovery CI/CD System https://www.ideas-engineering.io/blog/2018/07/disa
recovery-day

RECONMENDATIONS

Separation of services It is critical that you break apart your monolithic app into separate services such that they are isolated and abstracted from each other. https://www.computerweekly.com/feature/Deliver-quality-software-at-speed-with-CI-CD

DO CI GOOD Before you get to CD, make sure you do CI well. There are some foundational elements such as having consistent tooling, libraries and operating system configurations.

5 QUALITÄTSMERKMALE EINER CI/CD PIPELINE

...

6

FAZIT

Schwer weitgreifende Merkmale zu finden, dennoch ...

LISTE DER NOCH ZU ERLEDIGENDEN PUNKTE

LITERATUR

[1] Adam Wiggin. *The Twelve-Factor App*. [Online; accessed 20-January-2020]. 2014. URL: https://12factor.net.

[2] Sergejs Bobrovskis und Aleksejs Jurenoks. "A Survey of Continuous Integration, Continuous Delivery and Continuos Deployment." In: *BIR Workshops*. 2018, S. 314–322.

[3] Gene Brown Alanna Brown Nigel Kersten Dr. Nicole Forsgren Jez Humble. *2017 - State of DevOps*. 2017. URL: https://puppet.com/resources/report/2017-state-devops-report/.

[4] Divya Kumar und Krishn Mishra. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market". In: *Procedia Computer Science* 79 (Dez. 2016), S. 8–15. DOI: 10.1016/j.procs.2016.03.003.

[5] Winston W Royce. "Managing the development of large software systems: concepts and techniques". In: *Proceedings of the 9th international conference on Software Engineering*. 1987, S. 328–338.

[6] Mojtaba Shahin, Muhammad Ali Babar und Liming Zhu. "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices". In: *IEEE Access* 5 (2017), S. 3909–3943.