# Bemerkungen zu CI Seminararbeit von Daniel Wolfschmidt

# Einleitung

* Ich persönlich hätte eine andere Einleitung gewählt und würde auf die Probleme der Software-Entwicklung eingehen welche in Cloud/Umgebungen eingehen. Warum betrieben wir eigentlich CI? Was ist der Hintergrund? Und ganz wichtig, warum müssen wir neuestens überhaupt Code integrieren?

„Mittlerweile geht der Trend hin zu Microservices¹. Diese, bis auf die Schnittstellenbeschreibung, unabhängige Entwicklung der einzelnen Komponenten von Software, erlaubt eine wesentlich schnellere Erstellung von Software.“

* Keine Ahnung, ob das wirklich ein Trend ist. Microservices (MS) sind in der Cloud häufiger anzutreffen als in der Desktop-Entwicklung. Daher würde ich das nicht verallgemeinern. Für die Cloud könnte das schon stimmen.
* Dass die Entwicklung dadurch schneller ist, wage ich allerdings zu bezweifeln. Wer sich mit MS beschäftigt, der muss sich mit verteilten Systemen beschäftigen. Plötzlich tauchen Themen wie: Asynchronität, Resilient, Message-Queuing, Token-Based-Security, Live-Monitoring… auf. Dazu sind Teams mit anderen Fähigkeiten nötig

Auch die Einstellung der Nutzer hat sich verändert. Durch das schnelle Entwickeln von Patches und Updates, ist der Anwender zum Beta Tester von Software avanciert.

* Das würde ich so nicht schreiben. Der Anwender wird niemals zum Beta-Tester gemacht. Schnellere Releases sind eben einfach schnellere Releases. Nicht mehr. An der Qualität ändert sich nichts. Ganz im Gegenteil, Qualität bekommt einen ganz anderen Stellenwert. Aber das schreibst du später auch selbst in deiner Seminararbeit.

in weiterer Aspekt sind die neuen Vorgehensmodelle im (Software-)Projektmanagement. In der Vergangenheit war es gang und gäbe, das Wasserfall Modell zu verwenden. Dabei wird der Test in einer späten Projektphase durchge৒ührt

* Das Wasserfall-Model war nur ein Model. Viele andere fanden vorher schon Einkehr.
* Das Wasserfall-Model wurde interessanterweise auch einfach nur falsche verstanden und falsch großflächig kommuniziert. Legt man das Wasserfall-Model für jedes Release/Feature fest, so entsteht daraus ein agiles-Model. 😊

# 1.1 Begriffserklärung

Es geht hier also um das kollaborative Arbeiten in einem Team, insbesondere das Integrieren von Code in eine gemeinsame Code-Basis. Das heißt ferner, dass es eine Methodik ist die ৒ür einen einzelnen Entwickler kaum Bedeutung hat. Das ist auch einleuchtend, denn seinen eigenen Code in eben diesen zu integrieren, macht kaum Sinn.

* Da muss ich die leider wiedersprechen. Ich arbeite seit Jahren an vielen Projekten auch ganz alleine. Aber ohne CI ist vielen extrem schwierig. Mit der Meinung bin ich auch nicht alleine. Die Frage ist nämlich: Warum müssen wir überhaupt unseren Code integrireren?
  + Im Grunde geht es um das Zusammenstecken von Modulen, welche zur Entwicklungszeit getrennt entwickelt werden. Denn sobald Software aus einem einzigen Block besteht, muss nichts integriert werden.

Hierbei muss man auf eine sinnvolle Testtiefe achten

* Was ist eine sinnvolle Testtiefe?

Wenn man die Test-Pyramide³ zu Rate zieht, muss man darauf achten, dass alle Tests fertig sind bevor der nächste Entwickler sein Änderungen in die Code-Basis integriert. Deshalb ist es eventuell am Besten, wenn man während des initialen Builds nur Unittests aus৒ührt, und ein möglicherweise nur einmal am Tag laufender Build dann detaillierter testet.

* Das verstehe ich nicht. 😊

Zusammenfassend bleibt zu sagen dass mit Continuous Integration die Zusammenarbeit eines Entwicklerteams an einer gemeinsamen Code-Basis verbessert werden soll.

* Aus meiner Sicht geht es hier wenige um die Zusammenarbeit der Teams als um die Zusammenarbeit von Artefakten.

# 1.2 Ablauf von CI

Der Entwickler folgt diesem Prozess aus einem bestimmten Grund, nämlich entweder ein neues Feature zu implementieren oder bekannte Fehler in der Software zu beheben. Dies geschieht in diesem Schritt.

* Oder neue Tests oder neue Integration…

Der vorhergehende Schritt hat sich komplett auf das Implementieren von Code bzw. Code-

änderungen sowie das Implementieren und Ändern von Tests beschränkt. Diese müssen auch noch auf Fehlerfreiheit überprüft werden, bevor man sie in das SCM ein৒ügt. Die erste Kontrollinstanz ist nun der automatisierte Build auf der Entwicklermaschine. Darunter versteht man das Kompilieren und Linken der Quellen sowie das Aus৒ühren der zuvor geschriebenen automatisierten Tests. Hierbei beschränkt man sich zumeist auf UnitTests.

* Was dann aber häufig zu Brüchen bei den Integrationstests führt und somit CI/CD verhindert

11. **Give Feedbaࠪ about the outcome of the build**

Das bedeutet, dass alle anderen Entwickler die sich auf diesen Build stützen blockiert sind in ihrer Arbeit. Es heißt dann also so schnell wie möglich den Grund zu ਖ਼nden und dieses Problem zu beheben.

* Eine andere Möglichkeit sind Features-Branches. Ein Merge auf Master passiert dann nur nach erfolgreichem Integrationstests.
* Besser währe noch die Verwendung eines Package-Servers wie Nexus. Hierbei werden alle erfolgreich gebauten Artefakte in einem Repository verwaltet. Somit können alle Entwickler immer mit jeder beliebigen gebauten Version eines Builds arbeiten.

# 1.3 Gründe Continuous Integration einzusetzen

1. **Qalität steigern**

* Die gesteigerte Qualität ist nicht das oberste Ziel von CI. Vielmehr geht es um die Lösung eines technischen Problems. Die Integration von Codeteilen welche einzeln und isoliert entwickelt wurden.
* Dazu kommt die Sicherheit keine ungelösten Abhängigkeiten im Code zu haben. Du hast es bereits genannt. Build-Maschinen sind ganz anders als Entwickler-Maschinen.
* Code-Coverage ist nicht auf CI beschränkt und gibt es schon viele Jahrzehnte vor CI.

**Management Vorgaben**

* Sind aus meiner Erfahrung selten ein Grund. Da das Management in der Regel überhaupt nicht abschätzen kann, was das bedeutet. Meistens wird CI von der Entwicklung getrieben.

3. **Audit Trail**

* Verstehe nicht, was das mit CI zu tun hat.

4. **Sࠫnellerer und spontanerer Release mögliࠫ**

* Definitiv einer der wichtigen Gründe.

Das verkürzt den Restprozess bis zum Release und steigert auch das Vertrauen in den aktuellen Stand, da dieser regelmäßig und mehrfach getestet ist.

* Ich denke nicht, das durch die CI der Restprozess verkürzt wird. Die Arbeit wird dadurch ja nicht weniger. Aber ich bin definitiv bei dir wenn du sagst: „Das vertrauen wird durch CI gesteigert.“ Oh ja, sehr sogar. CI hat einen psychologischen Effekt auf das Team.

Wenn aus dem Markt nun ein besonders schwerwiegender Fehler gemeldet wird, kann man kurzfristig einen (Patch-)Release ansetzen und durch৒ühren

* Patches haben mit CI nichts zu tun. Dazu brauche ich kein CI. Patches gab es schon seit Beginn der Softwareentwicklung.

# 2.1 Kommerziell vs. Kostenlos

Bei einem kostenlosen Tool existiert dieser ਖ਼nanzielle Hintergrund nicht. Deshalb ist eine Tendenz zu erkennen, je größer das Unternehmen ist, desto wahrscheinlicher ist der Einsatz eines kommerziellen Tools.

* Amazon, Google, Youtube, Facebook, Github, Gitlab, Spotify, Zalando… setzen soweit ich das gesehen habe alle auf OpenSource.

**2.2 Hosted vs. On-Premise**

**Finanzielle Aspekte**

Für eher kleine Projekte ist es daher von Vorteil On-Premise zu verwenden, in großen Unternehmen und Projekten, wo eine Person sich nur um Build Automatisierung kümmern kann, ist dann wohl eher der eigene Server besser.

* Was ist mit Github und vorallem Gitlab?
  + Vollständige kostenlose CI/CD über Dockerhub…

**Naࠫvollziehbarkeit und Reproduzierbarkeit**

Ein anderer Aspekt ist vor allem ৒ür regulatorische Umgebungen wichtig. Hierbei muss es

möglich sein, nachzuvollziehen was alles Auswirkungen auf das Ergebnis des Builds hatte. Dazu gehört auch die Build Umgebung selbst. Diese ist jedoch bei vielen On-Premise Angeboten nicht eindeutig zu identiਖ਼zieren, da diese neue Features einfach live schalten. Auch ist es unmöglich einen Build mit einem bestimmten Stand der Build Automatisierung nochmals laufen zu lassen nach längerer Zeit. Man hat keine Chance, den alten Build Server wiederherzustellen.

* Leider stimmt das so nicht. Mit Vagrant kannst du selbstverständlich eine Build-Umgebung jederzeit herstellen.
* In meinen letzten Projekten haben wir das oft gemacht. Sobald die deine Anwendung paketierst (Docker), kannst du jeden beliebigen Build erneut ausführen. Docker-Scripte stellen dir genau die Umgebung her, welche du einmal definiert hat. Und zwar haargenau.