

Verteilte Systeme und Komponenten

Continuous Integration (CI)

Roland Gisler



Inhalt

- Was ist Continuous Integration?
- Die zehn Praktiken der CI
- Technische Umsetzung

Lernziele

- Sie können die Ziele der **C**ontinuous **I**ntegration (CI) erklären
- Sie kennen die zehn wesentlichen Praktiken der CI.

Continuous Integration

- Continuous Integration ist der Ausweg aus der «Integrationshölle»!



Continuous Integration (CI)

Hauptziele bei der Entwicklung von Software nach CI:

- Immer ein **lauffähiges** Produkt (Buildresultat) zu haben.
 - Es kann somit kontinuierlich getestet werden.
- Bei Fehlern jeder Art möglichst schnell ein Feedback zu erhalten.
 - Primär durch automatisierte Unit- und Integrationstests.
 - Aber auch durch Compiler, Classpath, statische Codeprüfung etc.
- Im Team parallel entwickeln zu können und dennoch...
 - den Überblick nicht zu verlieren.
 - den Integrationsaufwand zu minimieren.
 - über den aktuellen Zustand auf dem Laufenden zu sein.
- Moderne, **zeitgemässe**, agile Software-Entwicklung!
- Grundidee: Kent Beck (XP, JUnit) und Martin Fowler (Refactoring)

Die 10 Praktiken der CI

Übersicht: Die 10 Praktiken der CI – nach Martin Fowler

1. Einsatz eines (zentralen) Versionskontrollsystem.
2. Automatisierter Buildprozess.
3. Automatisierte Testfälle.
4. Alle Ändern den Quellcode auf dem Hauptzweig*.
5. Bei einer Änderung wird automatisch ein Build durchgeführt.
6. Der Buildprozess muss schnell sein.
7. Auf/mit Kopien der produktiven Umgebung testen.
8. Einfacher Zugriff auf aktuelle Buildartefakte.
9. Offensive Information über den aktuellen Zustand.
10. Automatisches Deployment*.

1 - Versionskontrollsystem

- Grundlagen: siehe Input → [EP 11 Versionskontrolle](#).
- Sämtliche Quellen-Artefakte welche für den vollständigen Build einer Software benötigt werden unterliegen der Versionskontrolle.
 - ➔ «Alles was für den Build benötigt wird, aber nichts was erneut gebaut (build) werden kann.»
- Nutzen Sie die Fähigkeiten des Versionskontrollsystems:
 1. Sinnvolle Commit-Kommentare vergeben.
 - Ideal: Mit Hinweis auf **Issue#** eines Issue Tracking Systems.
 2. Tagging – markieren von bestimmten Versionen.
 - Für die einfache, eindeutige Identifikation von Releases.
 3. Branches – Zweige für parallele Entwicklung.
 - Ermöglicht z.B. die parallele Weiterentwicklung, einfaches Bugfixing, (möglichst kurzlebige!) Feature-Branches etc.

2 - Automatisierter Buildprozess

- Grundlagen: siehe Input → [EP 12 Buildautomatisation](#)
- Build auf einer kontrollierten, «sauberen» Maschine durchführen.
 - Einsicht: Entwickler*innen-Maschinen sind **nie** sauber... 😊
- Ausschliesslich auf der Basis der aktuellen Quellen aus dem Versionskontrollsystem (VCS).
 - Dadurch stellt man z.B. sehr schnell und einfach fest, ob man vergessen hat etwas wesentliches einzuchecken.
- Inklusive Ausführung der Testfälle und QS-Metriken.
 - Laufen die Unit-Tests tatsächlich immer und überall?
 - Gibt es keine Seiteneffekte durch parallele Codeänderungen?
 - Ist die Codequalität gut genug?

3 - Automatisierte Testfälle

- Grundlagen: siehe Input → [EP 22 Automatisiertes Testing](#)
- Möglichst viel durch automatisierte Testfälle abdecken.
 - Primär Unit Tests, weil einfach und überall lauffähig.
 - Sekundär auch Integrationstests, z.B. Abhängig von Datenbank.
- Fehlerhafte oder nicht vollständige Implementationen sollen so schnell wie möglich aufdecken werden.
 - Bei Integrationstests auch häufig unerwartete Nebeneffekte.
- Bewährt haben sich auch Performance-Tests.
 - Wirkt sich ein neues Feature negativ auf die Performance aus?
- Primäres Ziel: Tests müssen immer laufen bzw. im Fehlerfall so schnell wie möglich wieder gefixt werden!
 - Gemeinsames Ziel für das **ganze Team**.

4 - Änderungen auf dem Hauptzweig des VCS

- Ursprünglich wurde gefordert, dass alle Entwickler*innen auf dem einzigen Hauptzweig (HEAD, trunk, master, main etc.) arbeiten.
 - Die Motivation ist, dass sämtliche Änderungen möglichst schnell (und kontinuierlich) in die **einzig**e Codebasis integriert werden.
 - Bei **grossen** Teams (welche entsprechend viele Änderungen produzieren) führt das aber zu sehr vielen «merges».
 - Mit moderneren VCS, welche «billige» Branches anbieten, begann man darum leichtfertig für jede Entwickler*in einen eigenen Branch zu eröffnen, um wieder autonomer arbeiten zu können.
- ➔ Das führt aber wieder zu einem sehr aufwändigen «BigBang», wenn diese Branches dann vor dem Release in den Hauptzweig gemerged werden (müssen)!!
- ➔ Man fällt wieder in die «Integrationshölle» zurück!



4.1 - Änderungen auf dem Hauptzweig – Ergänzung

- Es wurden verschiedene Branch-Konzepte entwickelt, die einen vernünftigen Kompromiss zwischen autonomer Arbeit, und trotzdem genügend häufiger Integration ermöglichen.
- Das populärste und bekannteste Prinzip: **GitFlow**
 - <https://nvie.com/posts/a-successful-git-branching-model/>
Mittlerweile von zahlreichen Tools integriert, um die (nicht immer trivialen) Abläufe zu automatisieren und vereinfachen.
- Empfehlung: **GitFlow** ist sehr gut, es lohnt sich aber oft, sich ein (ggf. vereinfachtes, siehe z.B. [GitHub-Flow](#)) und an das konkrete Projekt bzw. Team adaptiertes Konzept zu überlegen!
 - Teamgrösse, Projektgrösse, Modularisierung, Dynamik etc.
- Ziel: Häufige Integration, bei möglichst wenig Overhead!

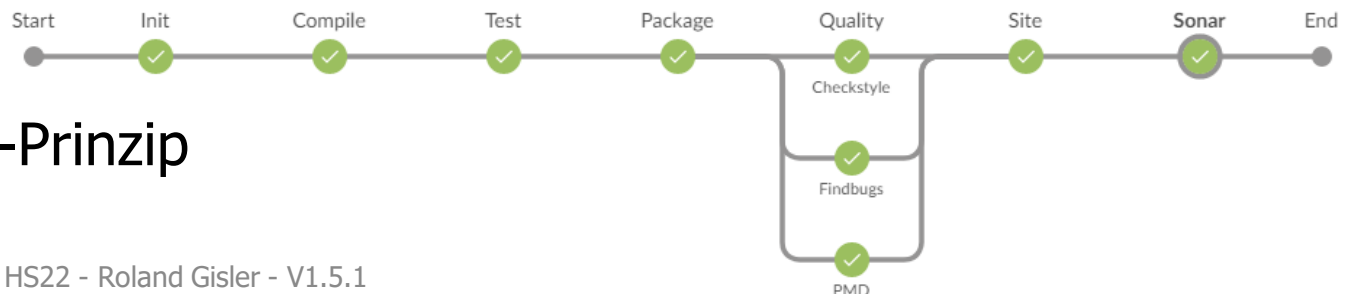
5 - Automatischer Build bei Änderungen



- Buildserver, siehe Input → [EP 14 Buildserver](#).
 - Voraussetzung: siehe Input → [EP 12 Buildautomatisation](#).
- Buildserver prüft regelmässig auf Veränderungen im Versionskontrollsystem (poll), bzw. wird vom SCM aktiv informiert (push).
 - Stellt er solche fest, werden die Quellen ausgecheckt und ein neuer Build gestartet.
- Alle Resultate des Build werden offensiv kommuniziert.
 - Erfolg, Testfälle, Laufzeit, Codechecks, Metriken – einfach alles!
- Ziel für das Team: Bricht ein Build (aus welchen Gründen auch immer), konzentriert man sich in erster Priorität und gemeinsam darauf, den Build wieder «grün» zu kriegen!
 - Proaktive Notifikation an alle beteiligten Entwickler*innen.

6 - Buildprozess muss schnell sein

- Je schneller die Entwickler*innen ein Feedback bekommen, dass etwas nicht mehr läuft, je besser!
- Natürlich muss ein Kompromiss gefunden werden – manche Tests benötigen mehr Zeit.
 - Werden somit kaum lokal durchgeführt.
 - Umso wichtiger ist es, diese im zentralen Build laufen zu lassen!
- Alternative: Zwei (oder mehr) gestaffelte Builds durchführen.
 - Schneller «continuous build für Feedback an Entwickler*innen.
 - Langsamer (weil umfangreicherer) «nightly build» über Nacht.
- Noch flexibler mit Build-Chains oder Pipelines: Sequenz von Builds.



- «fail-fast»-Prinzip

7 - Auf realer Umgebung testen

- Die zentrale Build- und Testumgebung sollte möglichst ähnlich zur produktiven Umgebung sein.
 - Ideal: Kopie der produktiven Umgebung → Teuer!
 - Alternative: Leichtgewichtige Virtualisierung z.B. mit Docker.
- Das umfasst z.B. folgende Punkte:
 - Hardwareausstattung, Betriebssystem, Laufzeitumgebung (Java etc.), Netzwerkzugriff (Kommunikation mit Drittsystemen)
 - Datenqualität und **Datenmenge**
 - Technologien wie Docker helfen uns hier extrem!
- In der Realität bei kleinen Systemen gut zu erreichen, bei grossen Systemen aber häufig viel zu teuer.
- Weitere Herausforderung: Datenschutz!

8 - Einfacher Zugriff auf Buildartefakte

- Sämtliche Buildresultate sollten jederzeit für eine weitere Nutzung zur Verfügung stehen.
- So dass einfach und schnell die neusten Versionen z.B. weiterführend getestet werden können.
 - Hat wiederum ein schnelleres Feedback zur Folge.
- Wird typisch über Buildserver erreicht, welche die Buildresultate selber archivieren können.
 - Achtung: Grosse Datenmengen möglich!
- Zusätzlich können die binären (ausführbaren) Artefakte zusätzlich noch in ein binäres Repository deployed.
 - z.B. Maven Repository, Sonar Nexus, Artifactory etc.
 - siehe Input → [EP 13 DependencyManagement](#)

9 - Offensive und offene Information











- Es gibt keine Geheimnisse!
- Für jede Entwickler*in ist jederzeit einsehbar welche Änderungen...
 - ...von wem und wann eingecheckt wurden.
 - ...von welchem Build erstmals erfasst wurden.
 - ...zu welchen Ergebnissen geführt haben (Build, Tests).
 - ...zu welchem Issue gehören.
 - ...welche Massnahmen getroffen wurden.
- Transparenz und Nachvollziehbarkeit: Offene Information nicht zur Kontrolle, sondern zur gemeinsamen Unterstützung!
 - Echtes «**collective code ownership**» als Ziel.
- Das Team hat ein **gemeinsames** Ziel:
Erfolgreiche Builds und eine möglichst fehlerfreie Software!

10 - Automatisches Deployment

- Wenn immer möglich sollte das Buildergebnis auch automatisch verteilt und installiert werden.
 - Installation auf einem repräsentativen Zielsystem.
 - Oder zumindest regelmässig automatisch aktualisieren.
 - z.B. Täglich, über Nacht etc.
 - Es macht wenig Sinn alle fünf Minuten eine Server-Applikation zu installieren, wenn ein manueller Test eine Stunde dauert...
 - Damit steht die aktuelle Software sofort wieder für weiterführende, (z.B. manuelle) Systemtests bereit.
 - Hat wiederum ein schnelleres Feedback zur Folge.
 - Vermeidet Meldung von Fehlern, die schon behoben sind.
- ➔ **DevOps**-Methoden, infrastructure-as-code, Container, Cloud...

CI im Modul VSK

CI im Modul Verteile Systeme und Komponenten (VSK)

-  1. Einsatz eines (zentralen) Versionskontrollsystem.
-  2. Automatisierter Buildprozess.
-  3. Automatisierte Testfälle.
-  4. Alle Ändern den Quellcode auf dem Hauptzweig.
-  5. Bei einer Änderung wird automatisch ein Build durchgeführt.
-  6. Der Buildprozess muss schnell sein.
-  7. Auf/mit Kopien der produktiven Umgebung testen.
-  8. Einfacher Zugriff auf aktuelle Buildartefakte.
-  9. Offensive Information über den aktuellen Zustand.
-  10. Automatisches Deployment.

Technische Umsetzung von CI

- ✓ ■ Moderne, zeitgemässe Entwicklungsumgebung (nicht nur IDE!)
- ✓ ■ Automatisierter, reproduzierbarer Buildprozess.
- ✓ ■ Buildserver-Technologie.
 - Ideal: Gegenseitige Vernetzung und Integration aller Tools.
- ✓ ■ Offene Information an alle Beteiligten.
 - Positiver Teamgeist, gemeinsames Ziel: Lauffähige Software!

Logger-Projekte: Jetzt aber!

- Projekte sollten immer «grün» (ohne Fehler baubar) sein.
- Zwischen- und Schlussabgabe: **GRÜN** ist **Pflicht und Ehre!**

S	W	Name	Last Version	# QG	# CS	# PMD	# SB	Zellenabdeckung	Branchabdeckung	Letzte Status	Letzte Dauer	Cause
🟢	✳	g00-loggerinterface	1.0.0-SNAPSHOT	6	1	0	0	0.0%	100.0%	14 Tage	1 Minute 3 Sekunden	🔍
🟢	✳	g01-demoapp	1.0.0-SNAPSHOT	4	3	0	1	0.0%	100.0%	5 Tage > 1 Monat	33 Sekunden	↑
🟢	✳	g01-logger	1.0.0-SNAPSHOT	151	87	12	14	57.75%	62.5%	5 Tage > 1.3 Monate	1 Minute 36 Sekunden	↑
🟢	✳	g01-stringpersistor	1.0.0-SNAPSHOT	60	50	6	2	84.62%	94.44%	5 Tage > 1.3 Monate	54 Sekunden	🔍
🟢	✳	g02-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	2.2 Tage > 1.3 Monate	12 Minuten	↑
🟢	✳	g02-logger	1.0.0-SNAPSHOT	959	535	272	18	0.11%	0.0%	2.2 Tage > 5.8 Tage	14 Minuten	🔍
🟡	✳	g02-stringpersistor	1.0.0-SNAPSHOT	50	35	1	8	78.79%	87.5%	5.8 Tage > 15 Tage	47 Sekunden	🔍
🟢	✳	g03-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	1.9 Stunden > 1.3 Monate	46 Sekunden	↑
🟢	✳	g03-logger	1.0.0-SNAPSHOT	25	17	0	1	2.38%	100.0%	1.9 Stunden > 17 Tage	1 Minute 41 Sekunden	🔍
🟢	🔗	g03-stringpersistor	1.0.0-SNAPSHOT	31	26	0	4	0.0%	0.0%	16 Tage > 1.3 Monate	53 Sekunden	🔍
🟢	✳	g04-demoapp	1.0.0-SNAPSHOT	4	2	0	1	64.86%	100.0%	4.9 Tage > 5 Tage	36 Sekunden	↑
🟢	✳	g04-logger	1.0.0-SNAPSHOT	56	40	5	1	0.0%	0.0%	4.9 Tage > 1.3 Monate	1 Minute 31 Sekunden	🔍
🟢	🔗	g04-stringpersistor	1.0.0-SNAPSHOT	0	0	0	0	100.0%	100.0%	13 Tage > 1.3 Monate	1 Minute 14 Sekunden	🔍
🔴	🔗	g05-demoapp	1.0.0-SNAPSHOT	-	-	-	-	82.76%	100.0%	1.9 Tage > 12 Tage	37 Sekunden	↑
🟢	🔗	g05-logger	1.0.0-SNAPSHOT	85	46	5	4	58.62%	31.25%	1.9 Tage > 4.1 Tage	11 Minuten	🔍
🟢	✳	g05-stringpersistor	1.0.0-SNAPSHOT	56	43	3	4	86.05%	87.5%	11 Tage > 1.3 Monate	1 Minute 4 Sekunden	🔍
🟢	✳	g06-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	13 Tage > 1.3 Monate	45 Sekunden	↑
🟢	✳	g06-logger	1.0.0-SNAPSHOT	11	5	1	1	5.13%	100.0%	13 Tage > 1.3 Monate	1 Minute 42 Sekunden	🔍
🟢	🔗	g06-stringpersistor	1.0.0-SNAPSHOT	13	9	0	3	0.0%	0.0%	14 Tage > 1.3 Monate	51 Sekunden	🔍
🟢	✳	g07-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	5.9 Tage > 1.3 Monate	54 Sekunden	↑
🟢	✳	g07-logger	1.0.0-SNAPSHOT	458	36	314	9	0.14%	0.0%	5.9 Tage > 1.3 Monate	2 Minuten 19 Sekunden	🔍
🟢	✳	g07-stringpersistor	1.0.0-SNAPSHOT	27	21	0	1	80.3%	76.67%	12 Tage > 1.3 Monate	1 Minute 10 Sekunden	🔍
🟢	✳	g08-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	22 Stunden > 1.3 Monate	44 Sekunden	↑
🟢	✳	g08-logger	1.0.0-SNAPSHOT	85	52	3	2	3.75%	0.0%	22 Stunden > 20 Tage	1 Minute 46 Sekunden	🔍
🟢	🔗	g08-stringpersistor	1.0.0-SNAPSHOT	58	39	1	8	76.47%	78.57%	13 Tage > 13 Tage	47 Sekunden	🔍
🟢	✳	g09-demoapp	1.0.0-SNAPSHOT	0	0	0	0	85.71%	100.0%	2.3 Stunden > 1.3 Monate	44 Sekunden	↑
🟢	✳	g09-logger	1.0.0-SNAPSHOT	48	28	3	10	0.97%	0.0%	2.3 Stunden > 2.5 Tage	1 Minute 39 Sekunden	🔍
🟢	✳	g09-stringpersistor	1.0.0-SNAPSHOT	107	88	4	10	16.83%	10.0%	7.2 Tage > 1.3 Monate	1 Minute 2 Sekunden	🔍
🟢	✳	g10-demoapp	1.0.0-SNAPSHOT	1	0	0	1	0.0%	0.0%	7.1 Tage > 19 Tage	42 Sekunden	↑
🔴	🔗	g10-logger	1.0.0-SNAPSHOT	-	-	-	-	62.03%	60.0%	3.8 Stunden > 7.1 Tage	2 Minuten 3 Sekunden	🔍
🟢	✳	g10-stringpersistor	1.0.0-SNAPSHOT	13	11	1	1	88.57%	100.0%	13 Tage > 19 Tage	1 Minute 5 Sekunden	🔍
🟢	✳	g11-demoapp	1.0.0-SNAPSHOT	2	2	0	0	57.14%	100.0%	4.8 Tage > 1.3 Monate	37 Sekunden	↑
🟢	✳	g11-logger	1.0.0-SNAPSHOT	82	60	1	1	43.94%	25.0%	4.8 Tage > 1.3 Monate	1 Minute 34 Sekunden	🔍
🟢	🔗	g11-stringpersistor	1.0.0-SNAPSHOT	29	26	0	2	80.0%	83.33%	15 Tage > 1.3 Monate	1 Minute 7 Sekunden	🔍

- Stichprobe: Mittwoch, 2. November 2022 – **ok!**

Zusammenfassung

- Continuous Integration nach Martin Fowler:
Anwendung von zehn konkreten Praktiken.
- CI darf/soll niemals Selbstzweck sein!
- Pragmatische Anpassung an die Bedürfnisse des jeweiligen Projektes und Teams.
- Gewinn und Nutzen von CI steht im Vordergrund:
Kollektive Bearbeitung des Quellcodes, immer ein lauffähiges Produkt, Nachvollziehbarkeit der Entwicklung.
- Grundlage für zeitgemässe iterative und inkrementelle Entwicklung von Software.

Fragen?