

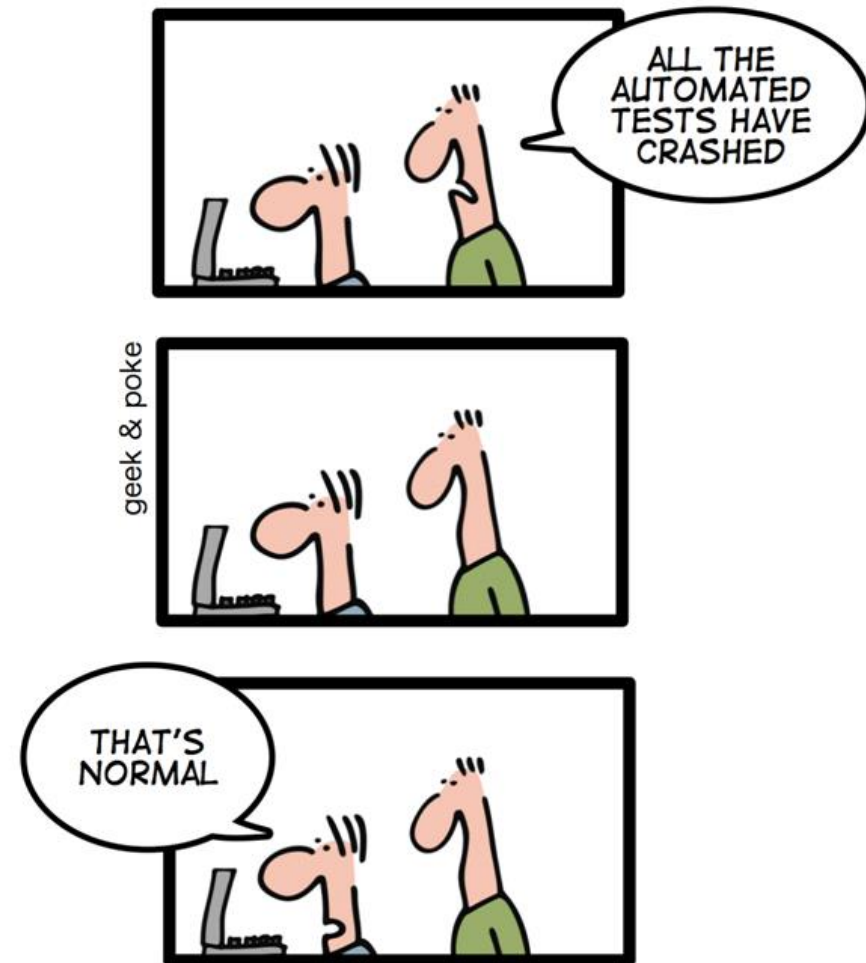


# Continuous Integration (CI)

HAW Hamburg / Fachbereich Informatik

Tim Luecke

([Tim.Luecke@haw-hamburg.de](mailto:Tim.Luecke@haw-hamburg.de))



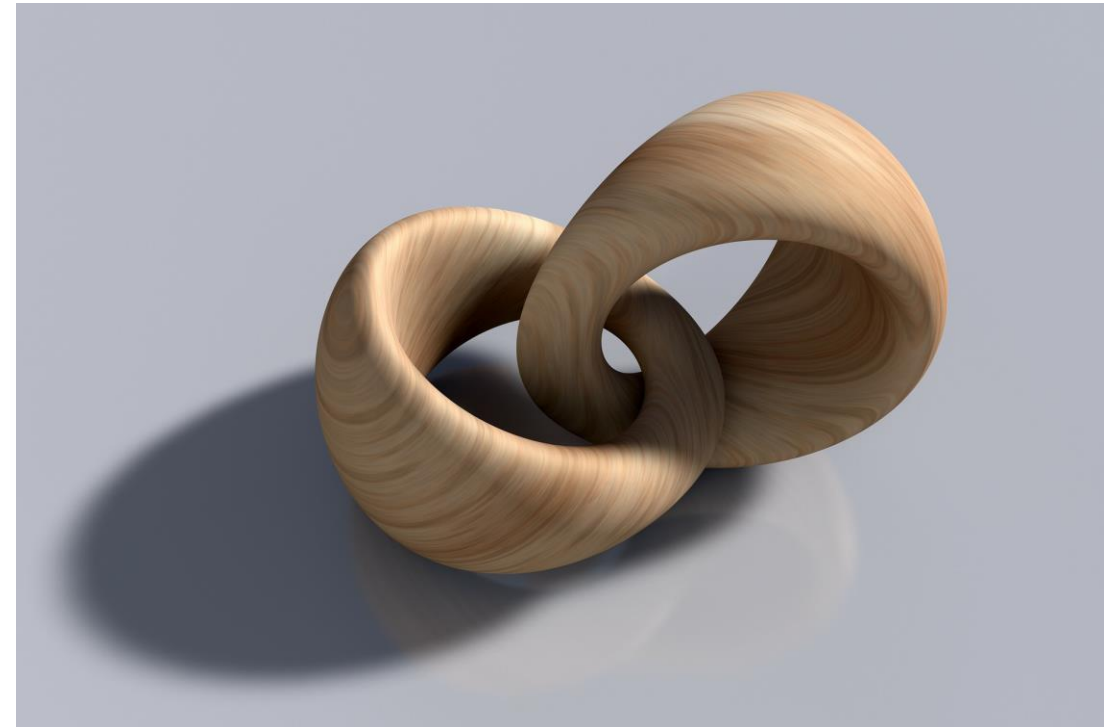


*"The key to fixing problems quickly is finding them quickly"*

*(Martin Fowler)*

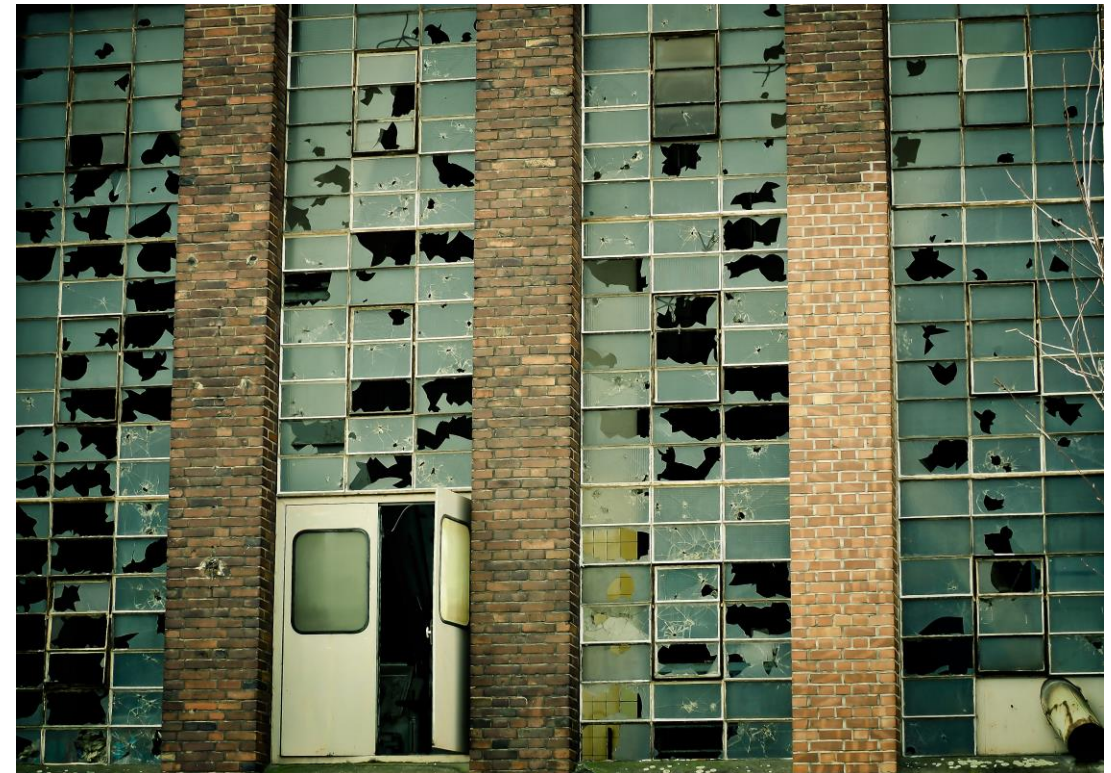
# Was ist Continuous Integration?

- Eine der 12 ursprünglichen Praktiken des „*Extreme Programming*“ („*Agile*“ -> *später*)
- **Regelmäßige** Integration von Code in die Mainline
  - „Mainline“: z.B. Masterbranch, Developerbranch, Releasebranch, ...
- Die Mainline soll **fehlerfrei** gehalten werden



# Motivation

- Fehler werden schnell gefunden
- Fehler lassen sich schneller beheben
- Es sind insgesamt weniger Fehler im System
  - Je mehr Fehler im System sind, desto schwieriger ist es einen einzelnen zu beheben
  - Psychologische Auswirkungen („[Broken-Windows-Theorie](#)“)
- Transparenz über Qualitativen Zustand des Systems
- Grundlage für Continuous Deployment





# Vorraussetzung für CI

## An die Technik...

- Source Control System
- Automatisierte Builds
- Automatisierte Testdurchführung
- Schnelle Builds



## An den Entwicklungsprozess...

- Mainline immer fehlerfrei halten
- Selbsttestenden Code schreiben
- Aufteilung größerer Aufgaben
- Mindestens einmal am Tag committen
- Kommunikationsbereitschaft der Entwickler untereinander









# Wie funktioniert CI?

1. Holen einer lokalen Kopie des Codes  
**git clone**
2. Programmieren einer kleinen Erweiterung  
(nur wenige Stunden)
3. Lokales Bauen des Systems und Durchführen der Testfälle für das gesamte System  
**./gradlew build**
4. Update auf die aktuelle Version  
**git pull**  
(da in der Zwischenzeit andere Entwickler Änderungen eingereicht haben können)
5. Beheben aller Konflikte und erneutes Durchführen der Testfälle  
**./gradlew build**
6. Einchecken des integrierten Codes  
**git add; git commit; git push**

7. Integrationsserver baut Mainline und führt Testfälle durch

Status	Pipeline	Commit	Stages
 passed	#22839 by  latest	 master -> 5434b435 gradle update, moved java in...	

8. Integrationsserver macht gebautes System zugänglich  
(Deployment in gitlab konfigurieren)
9. Integrationssserver benachrichtigt Committer über Erfolg des Buildes





# CI Produkte

- GitLab CI  
<https://gitlab.informatik.haw-hamburg.de>
- Jenkins  
<http://jenkins-ci.org/>
- Travis-CI  
<https://travis-ci.org/>
- Atlassian Bamboo:  
<https://de.atlassian.com/software/bamboo>
- JetBrains TeamCity:  
<http://www.jetbrains.com/teamcity/>
- ...



# Demo

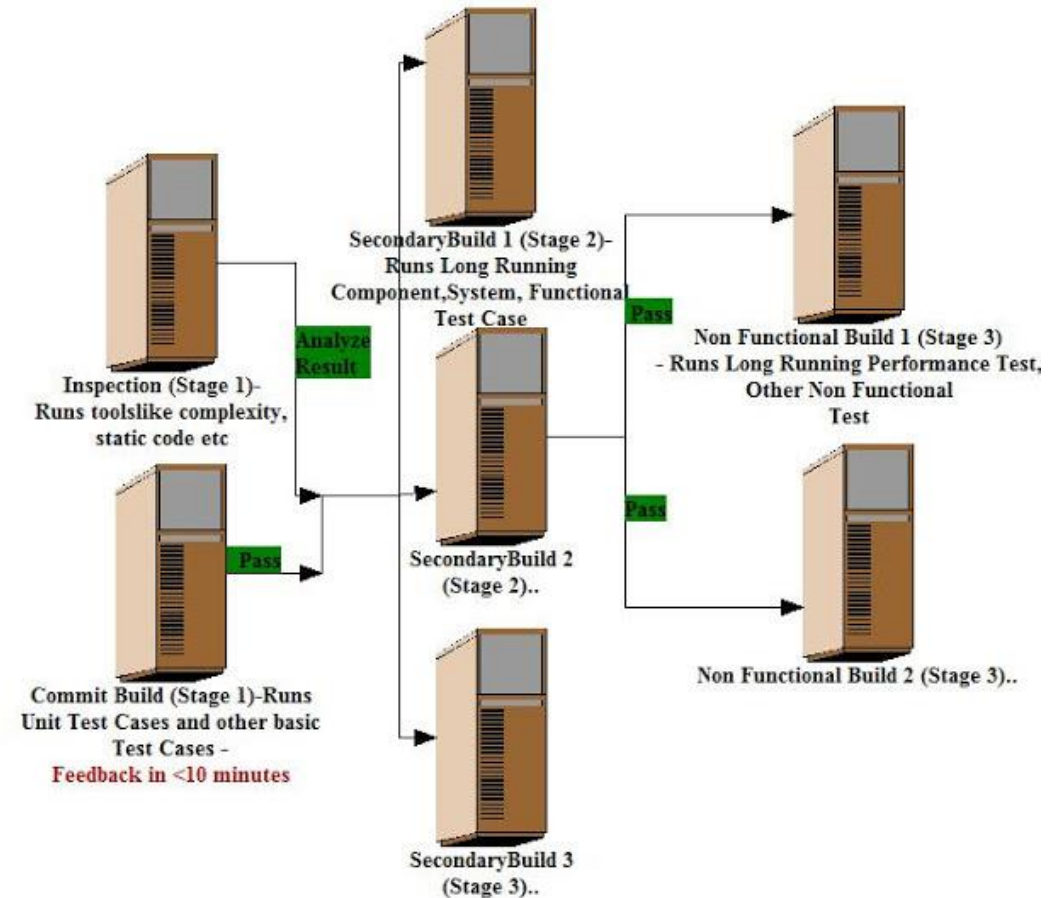
<https://gitlab.informatik.haw-hamburg.de/acp343/demo-project>





# Mögliche Probleme

- **Problem:** Fehlerfreiheit der Mainline nicht sichergestellt
- **Lösung:**
  - Nicht direkt auf die Mainline committen (Branch(es) anlegen -> Branch-Strategien)
  - Dezentrale Ansätze wie GIT (jeder Entwickler hat ein eigenes Repository)
  - Oder fehlerhafte Builds werden zunächst „zwischencommitted“ und CI-Server überprüft; falls erfolgreich -> „finales“ commit in das Repository
  - einstellbar in manchen Produkten
- **Problem:** Build dauert zu lange
- **Lösung:**
  - Staged Builds (Build pipeline)
  - Projekt sinnvoll gliedern (Komponenten!)
  - nicht alle Tests immer durchführen (Performancetests z.B. seltener -> am Wochenende)



<http://www.thinkinginagile.com/2011/06/continuous-integration-what-is-staged.html>



# Zusammenfassung

- **Continuous Integration (CI)** macht Integrationsaufwand kontrollierbar
- **Probleme** werden sofort sichtbar und können direkt behoben werden
- Erfordert (neben der Technik) viel **Disziplin**
- Zunehmend werden auch Deployments/Installation bzw. der Aufbau ganzer (Test-) Umgebungen automatisiert  
(Stichwort **Continuous Deployment/Continuous Delivery**:  
[http://de.wikipedia.org/wiki/Continuous\\_Delivery](http://de.wikipedia.org/wiki/Continuous_Delivery))  
-> später, zudem AI-Vorlesung, Vertiefung durch WP-Angebote