Hochschule
für Wirtschaft und Gesellschaft
Ludwigshafen

Aktuelle Themen der IT

Agenda

- Besprechung TakeHome (SimpleCalculator + Library)
- Wiederholung GitHub
- Agiles Entwickeln
- Clean Code Development
- Test Driven Development
- Coding Dojo

Take-Home

Aufgabe: Taschenrechner

1. Identifiziert zu testende Module
2. Definiert Testfälle für die Module
3. Implementiert die Testfälle mit JUnit
4. Identifiziert zusätzliche Fehlerquellen im Programmcode
5. Definiert sinnvolle Tests für die identifizierten Fehlerquellen oder refactort den Quellcode

Aufgabe: Bibliothek

1. Identifizieren der Module
2. Definition der Testfälle
3. Implementierung der Tests mit JUnit
4. Identifizierung weiterer Fehlerquellen (logische/strukturelle Fehler, etc.)
5. Refactoring des Quellcodes

Wiederholung GitHub

GitHub

GitHub 

- Basiert auf dem Open Source Quellcode-Verwaltungssystem Git
- Git dient zur Versionsverwaltung von Quelltext und unterstützt den Softwareentwicklungs-Prozess
- GitHub ermöglicht die Kollaboration von Entwicklern an Projekten durch die darunter liegende Plattform
- Verteilte Software-Repositories ermöglichen die gemeinschaftliche Arbeit an Projekten
- Viele weitere nützliche Funktionen (Pages, Doku, Issue Tracking, Release-Mgmt, etc.)

Git und GitHub sind aktuell die meistgenutzten Quellcode-Verwaltungssysteme in der Softwareentwicklung

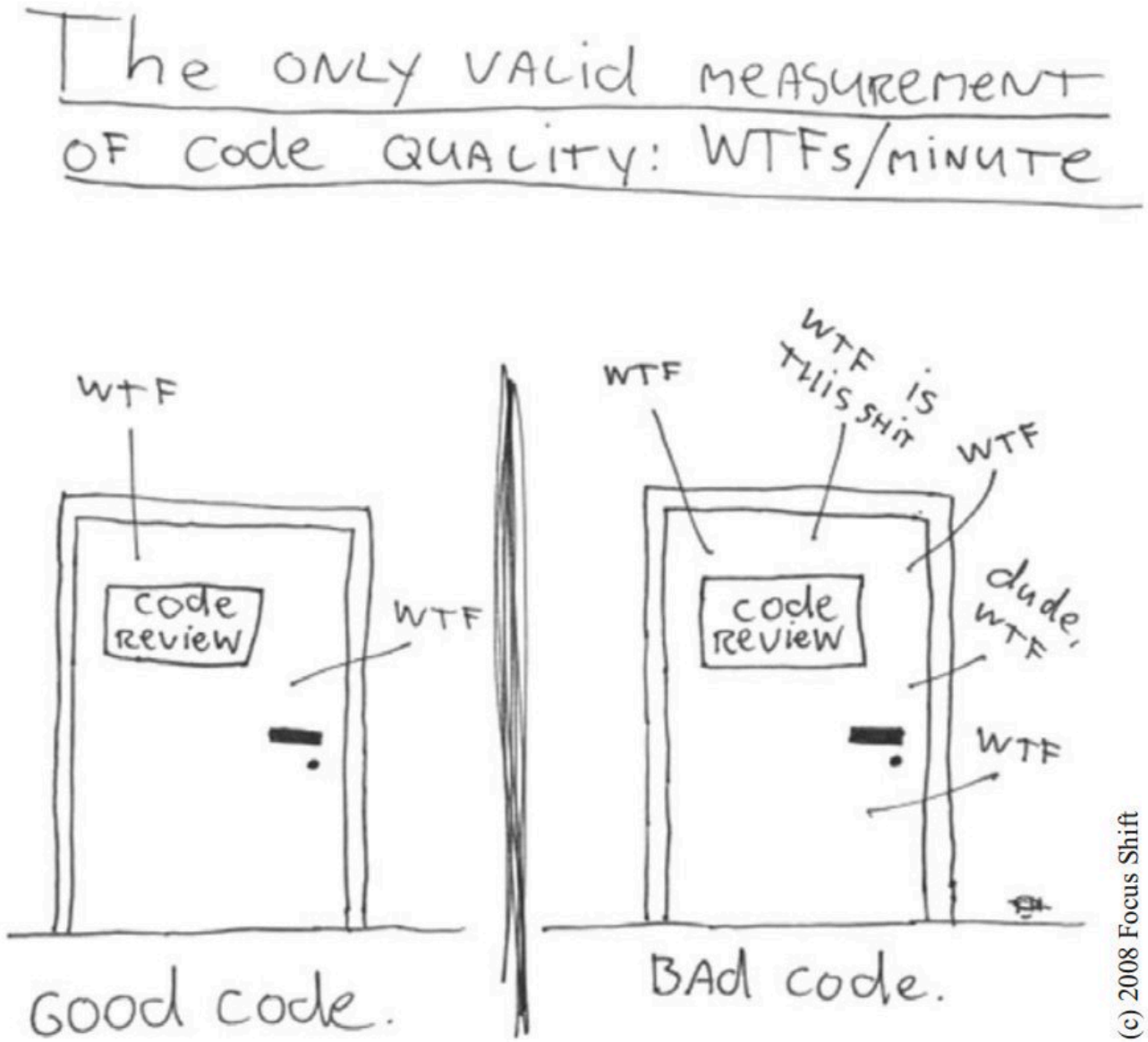
GitHub

GitHub 

Basic Commands:

- $ git init // Initialisieren eines lokalen Git Repositories
- $ git add <file> // Dateien zum Index hinzufügen
- $ git status // Status des Working Tree
- $ git commit // Änderungen auf den Index anwenden
- $ git push // auf remote Repository pushen
- $ git pull // aktuellen Stand vom remote Repository holen
- $ git clone // Repository klonen

Clean Code Development



Reproduced with the kind permission of Thom Holwerda.
http://www.osnews.com/story/19266/WTFs_m

Clean Code Development

Uncle Bob: Robert C. Martin („Clean Code - A Handbook for Agile Software Craftsmanship“)

Sauberer Quellcode: für jedermann intuitiv und eindeutig verständlich

Herausforderung:

- unklare oder widersprechende Anforderungen
- fehlende Erfahrung der Entwickler
- Mangel an Disziplin
- Aufwand des Refactorings

Software-Craftsmanship (Bewegung mit starkem Fokus auf qualitative Softwareentwicklung)

Prinzipien des CCD

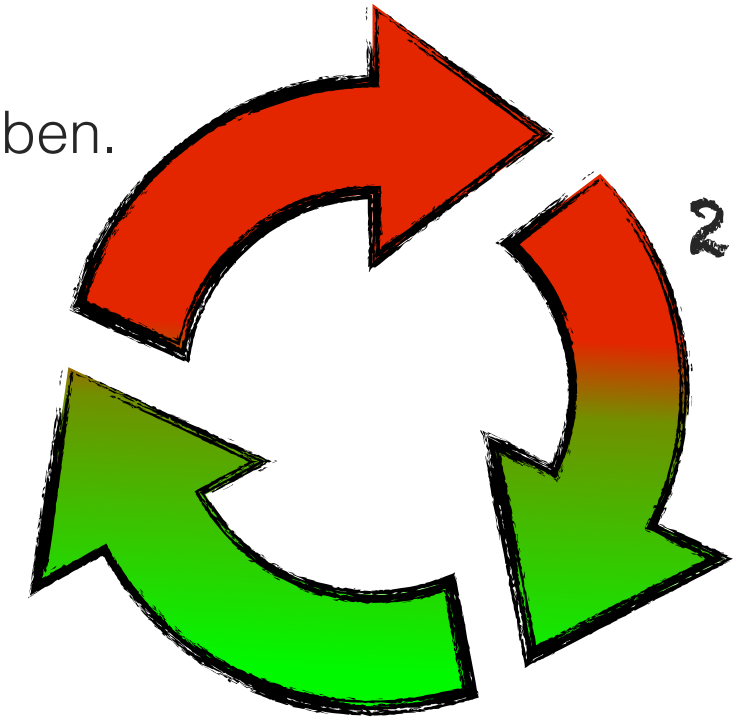
- Don’t Repeat Yourself (DRY)
- Keep it Simple, Stupid! (KISS)
- „Premature Optimization is the root of all evil“ (Donald E. Knuth)
- Tell, don’t ask (TDA)
- Separation of Concerns
- Single Responsibility Principle (SRP)
- …

Test Driven Development

Test Driven Development

1. Test für neuen Code schreiben.

   Test schlägt fehlt!



2. Code zur Erfüllung des Tests schreiben.

   „Do the simplest thing you can!“

3. Vergewissere dich ob der Test grün ist.

   Refactor den Quellcode!

Ziel

- Fehlerfreien Code direkt beim ersten Entwurf zu schreiben
- „Code Once“
- Hohe Test Coverage mit initialen Implementierung
- Reduktion Einarbeitungskosten bei nachträglicher Test-Implementierung
- Klare Definition der Funktionalität einzelner Bausteine/Module
- saubere testbare Architektur by Design
- Konsequentes Refactoring: wenig Redundanz/toter Code -> Clean Code
- Konzentration auf das Wesentliche

Hands-On

Coding Dojo

Coding Dojo?

- Dojo (= jap. Raum zum Training, bekannt aus vielen jap. Kampfsportarten)
- gemeinsames Verbessern/Training im Vordergrund
- Kein Wettkampf
- „Katas“ bezeichnen die vollzogenen Aufgaben (= jap. Form/Figur)
- unterschiedlich Schwerpunkte (Function, Class, Library, Application, Achitecture, Agility Katas)
- Constraints: Zusätzliche Herausforderungen im Dojo

Dojo Setup

- Teams of Two - PairProgramming
- Team arbeitet an einem PC und einer Lösung
- Test Driven Development
- Constraints limitieren die Vorgehensweise
- time boxed: jede Iteration hat fixe Dauer
- Hard Reset nach jeder Iteration
- Kurzes Recap nach jeder Iteration in der Gruppe
- Gleichbleibende Aufgabenstellung in allen Iterationen: Kata

Kata: Game of Life

Conway's Game of Life

- Population eines Feldes
- undefiniert großes Spielfeld
- beliebiger Anfangszustand

- < 2 lebende Nachbarn: Zelle stirbt wegen Unterpopulation
- > 3 lebende Nachbarn: Zelle stirbt wegen Überpopulation
- 2,3 lebende Nachbarn: Zelle bleibt am Leben
- jede tote Zelle mit exakt 3 lebenden Nachbarn: erwacht zum Leben

Iteration 1

Recap

Iteration 1

Iteration 2

Recap

Iteration 2

Constraint:

No voids!

Iteration 3

Recap

Iteration 3

Dojo Recap

Conway's Game of Life

Stanford: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2001-02/cellular-automata/index.html

GitHub Sample:https://github.com/fabricejeannet/kataGameOfLife/blob/master/src/main/java/GameOfLife.java

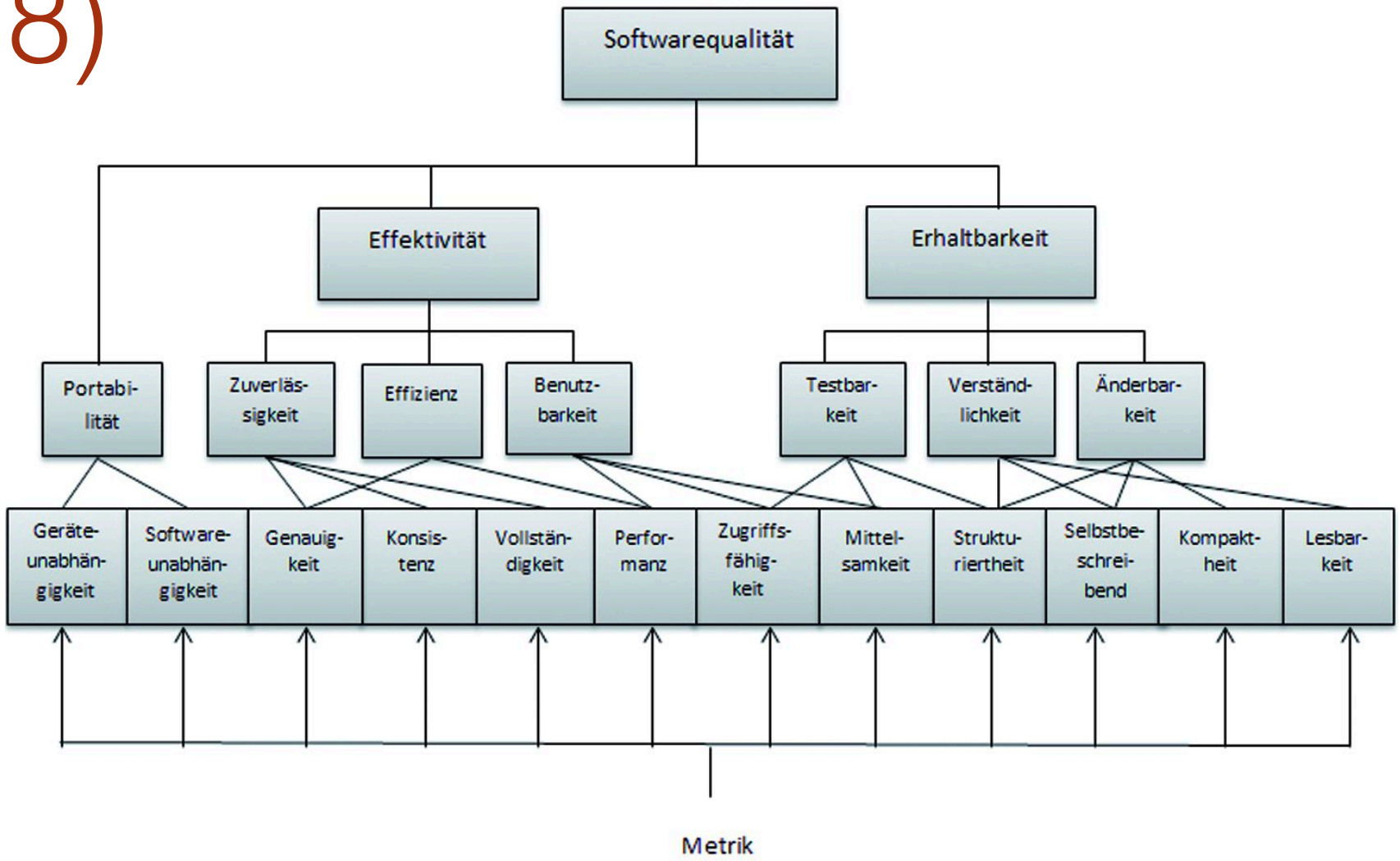
Live Demo: https://playgameoflife.com/

Backup

Merkmale guter Software

• Funktionalität	Interoperabilität, Richtigkeit, Angemessenheit, Sicherheit
• Zuverlässigkeit	Reife, Fehlertoleranz, Wiederherstellbarkeit
• Benutzbarkeit	Bedienbarkeit, Erkennbarkeit, Verständlichkeit, Erwartungstreue
• Effizienz	Zeitverhalten, Verbrauchsverhalten, Stabilität
• Änder-/ Modifizierbarkeit	Analysierbarkeit, Prüfbarkeit
• Übertragbarkeit	Installierbarkeit, Anpassbarkeit

Qualitätseigenschaften nach Boehm (1978)



Qualität durch Qualitätsmanagement

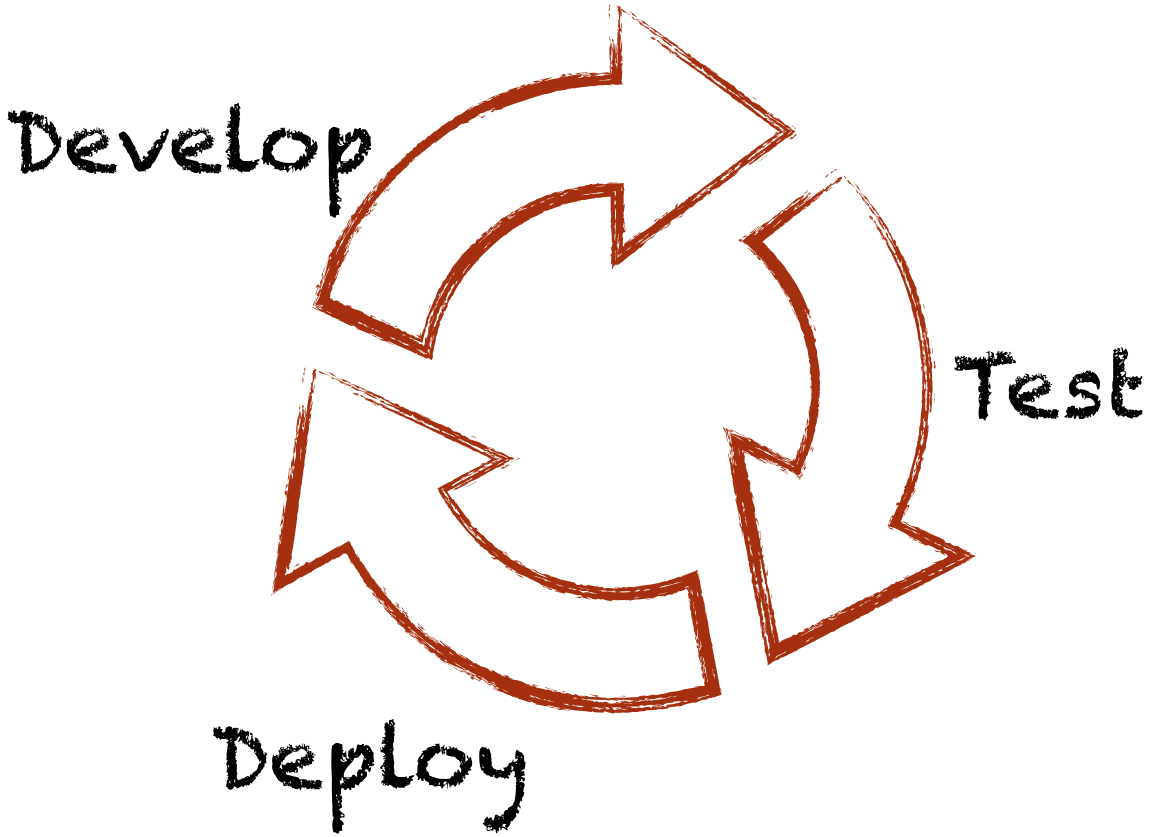
1. Qualitätsplanung (z.B. Review im Zeitplan)
2. Qualitätslenkung (Überwachung und Steuerung)
3. Qualitätssicherung (Prüfmethoden & Tests)
4. Qualitätsverbesserung & -prüfung (Produkte & Prozess)

Prinzipien der Qualitätssicherung

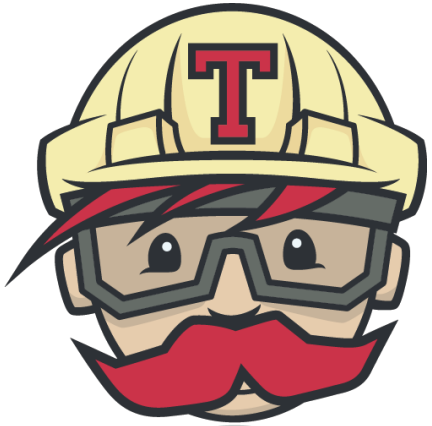
- Quantitative Qualitätssicherung (schwierig)

  Zahlenmässiges Erfassen / Metriken
- Entwicklungsbegleitende QS
- Unabhängige QS
- Frühzeitige Fehlererkennung
- Max. konstruktive QS
  - zeitlicher Faktor (im Vorfeld)
  - Methoden & Werkzeuge
  - Prüfung / Test
  - Bewertung / Resultate

Continuous Integration



Travis



- Automatisierte Testausführung
- Integriert in GitHub
- Verteiltes Integrationssystem
- baut und testes Softwareänderungen vor Merge/Deployment
- einfache Konfiguration und Einrichtung
- Integration von Sub-Anbietern (Coverage Analyzers, Statische Code Analyse)
- Open Source

Jenkins

- Automationssystem für CI/CD
- Komplexere Installation
- Vielzahl an Erweiterungen
- State of the Art for CI/CD
- Verteilte Architektur (spezialisierte Knoten für dedizierte Aufgaben konfigurierbar)
- Integration in Git/GitHub und viele andere Quellcode-Verwaltungssysteme
- Open Source