
JOHANNES KEPLER
UNIVERSITY LINZ

Praktikum Software Engineering

Johannes Bräuer

Einheit 0 - Einführung, Vorbesprechung

JOHANNES KEPLER
UNIVERSITY LINZ

Agenda

- Organisatorisches
- Gruppeneinteilung
- GitHub Zugang
- Aufgabenverteilung für Workshop am 13.10.

Inhalt des Praktikums

- **Agile Entwicklung eines Softwareproduktes im Team**
  - ➔ Spezifizieren, Planen und Entwerfen eines Produktes
- **Implementierung, Testen und Qualitätssicherung**
- **Arbeiten mit Software Engineering Werkzeugen**
  - ➔ Versionsmanagement (Git)
  - ➔ Projektmanagement (GitHub, Clockify)
  - ➔ Buildmanagement (Maven, GitHub Actions)
- **Planen von Sprints und Releases**
- **Erstellen einer Systemdokumentation**

Thema: Digital Driver's Logbook

- **Entwicklung einer Applikation zur Fahrtenverwaltung**
  - ➜ Erstellung und Verwaltung von Fahrten
  - ➜ Speichern und Backup
  - ➜ Filtern und Sortieren
  - ➜ Analyse und Statistiken
- **Technologien:** Java mit geeigneten Bibliotheken, entweder als Desktop-Applikation, Webapplikation oder mobile Applikation*

  * **Alle** Teammitglieder müssen mit der ausgewählten Technologie vertraut sein!

Organisation

- **Arbeiten in Teams zu 3 Studierenden**
  - ➜ Grundlegende Requirements als Vorgabe
- **Aufwand und Aufgaben müssen gleich verteilt werden**
  - ➜ Aufwand: 6 ECTS (~ 150 Arbeitsstunden)
- **Dringende Empfehlung**
  - ➜ Absolvierung der LVAs Soft1, Soft2, SE

<u>Jedes Teammitglied</u> hat Implementierungsaufgaben zu übernehmen – auf annähernd gleiche Verteilung achten!

Agile Software Entwicklung

- **Entwicklung in definierten Iterationen (Sprints)**
  ➜ 1 Woche bis max. 1 Monat
- **Priorisierung der Requirements**
  ➜ Team entscheidet welche Requirements im jeweiligen Sprint umgesetzt werden
- **Ergebnis eines Sprints = Neue Version des Produktes**
- **Hohes Maß an Selbstorganisation**

Zeitplan

- **Die Applikation wird in drei Sprints entwickelt**
  - ➜ Release 1: Di, 14. November (12.00 Uhr) - `release-0.1.0`
  - ○ Präsentation: Fr, 17. November *) (remote)
  - ➜ Release 2: Di, 12. Dezember (12.00 Uhr) - `release-0.2.0`
  - ○ Präsentation: Fr, 15. Dezember
  - ➜ Release 3: Di, 16. Jänner (12.00 Uhr) - `release-0.3.0`
  - ○ Präsentation: Fr, 19. Jänner
  - ➜ Finale Abgabe: Fr, 9. Februar (12.00 Uhr) - `release-1.0.0`
- **Abgabe <u>pro Sprint</u>:**
  - ➜ Branch in GitHub mit allen Dokumenten + Code

Präsenztermine - Sprint Planning Meetings

- **Anforderungsworkshop, Teampräsentationen (13. Oktober)**
- **Präsentation Prototype, JavaFX (27. Oktober)**
- **3 Sprint Planning Meetings (Release 1-3)**
  - ➔ Anwesenheitspflicht des gesamten Teams
  - ➔ 10-15 Minuten Präsentation (Folien-Template)
  - ➔ Jedes Teammitglied präsentiert
  - ➔ Diskussion, Status, Next Steps, …
- **1 individueller Termine pro Team**
  - ➔ Feedback & Fragen (30 Minuten)
  - ➔ Anwesenheitspflicht des gesamten Teams

Präsentation Prototype - 27. Oktober

- **Fokus: UI Prototyp**
- **Deliverables**
  - ➔ UI Prototype
  - ➔ Lauffähige Builds in GitHub Actions
  - ➔ Grobkonzept zum Aufbau der Applikation
  - ➔ Planung Release 1
  - ➔ Live Demo des UI Prototyps

Sprint 1 - 17. November

- **Fokus: Prototyp Implementierung und Unit Tests**
- **Deliverables**
  - ➔ UML Klassendiagramm mit den wichtigsten Klassen (Klassennamen, Hierarchien, Methoden…) mit UML Tool!
  - ➔ Prototyp Implementierung
    - Erste lauffähige Version des UIs
    - Teile der Features funktionsfähig (ca. 1/3 des Funktionsumfangs)
  - ➔ Unit Tests für einzelne Klassen
  - ➔ Planung Release 2
  - ➔ Live Demo der Applikation

Sprint 2 - 15. Dezember

- **Fokus: Dokumentation & Code Qualität**
- **Deliverables**
  - ➜ Erweiterte/aktualisierte UML Diagramme
  - ➜ Erweiterte Unit Tests
  - ➜ Implementierung
    - Ca. 2/3 des Funktionsumfangs umgesetzt
  - ➜ Analyse der Codequalität mit SonarQube, PMD, Findbugs, etc.
  - ➜ Erste Version der Projektdokumentation
  - ➜ Planung Release 3
  - ➜ Live Demo der Applikation

Sprint 3 - 19. Jänner

- **Fokus: Stabilität und Projektfinalisierung**
- **Deliverables**
  - ➔ Implementierung
    - Alle Features umgesetzt
  - ➔ Testplan für Sonderfallbehandlungen
  - ➔ GitHub Dokumentation (Installationsanleitung, usw.)
  - ➔ Javadoc für wichtige Klassen, Interfaces und Methoden
  - ➔ Live Demo der Applikation

Finale Abgabe - 9. Februar 2024

- **Deliverables**
  - ➔ Finale Projektdokumentation
  - ➔ Lauffähige, finale Version der Applikation
  - ➔ GitHub Dokumentation (Installationsanleitung, usw.)
  - ➔ Individueller Erfahrungsbericht

Beurteilung

- **Kriterien für die Beurteilung**
  - ➔ Funktionsumfang des Produktes
  - ➔ Externe Qualität des Produktes
    - ○ Stabilität, Performanz, User Interface
  - ➔ Interne Qualität des Produktes
    - ○ Qualität des Entwurfs, Implementierung, Dokumentation
  - ➔ Umfang und Qualität der Unit Tests
  - ➔ Präsentationen

Werkzeuge

- **GitHub / GitHub Actions**
  - ➔ Sourcecode Verwaltung
  - ➔ Verwaltung von Requirements
  - ➔ Buildmanagement (Continuous Integration)
- **Clockify**
  - ➔ Zeitaufzeichnung
- **Maven**
  - ➔ Buildmanagement
- **UML Editor / UI Prototyping Tool**
- **Code Qualität: Statische Code Analyse Tools**

Projektmanagement mit GitHub

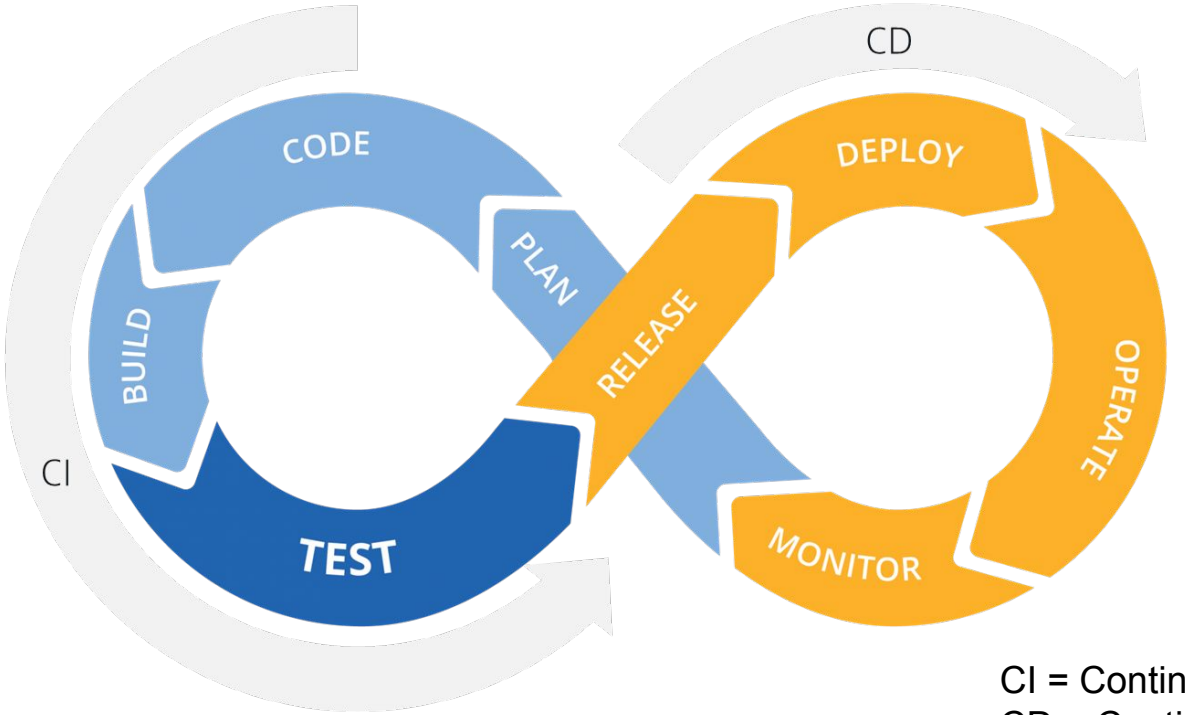
- **Umsetzungsdetails in GitHub**
  - ➔ Requirements werden als Issues beschrieben
  - ➔ Für jeden Sprint: Issues, Tasks, Bugs, etc. anlegen
  - ➔ Verantwortungen und Aufwände eintragen - Verantwortlichkeit definiert, wer Sourcecode schreibt (Code + Unit Tests)
- **Sprint Planung in GitHub**
  - ➔ Am Ende jedes Sprints müssen die jeweiligen Issues geschlossen werden und den Releases zugeordnet sein

Sourcecodeverwaltung mit Git

- **GitHub für die Code- und Dokumentationsverwaltung**
  - ➔ Commit mindestens 1x pro Woche (von jedem Teammitglied)
  - ➔ Bei jedem commit immer die jeweilige Issue-ID eintragen
- **Auf Codequalität achten**
  - ➔ Probleme die nicht gefixt werden, entsprechend dokumentieren
- **Abgabe jedes Releases als eigener Branch in GitHub**

`(release-0.1.0, release-0.2.0, release-0.3.0, release-1.0.0)`

Continuous Integration (CI) mit GitHub Actions



CI = Continuous Integration
CD = Continuous Delivery

GitHub Actions

- GitHub Actions ist eine Cloud-basierte CI Platform
- Unterstützt gängige Programmiersprachen: Nodes.js, Python, **Java**, Ruby, PHP, Go, Rust, .NET, etc.
- Baut die Software für unterschiedliche Betriebssysteme

- GitHub Actions ist Event-basiert, was bedeutet dass bestimmte Events die Ausführung eines Workflows anstoßen können.
  - ➜ Ein Event startet einen Job, der aus Steps zusammengesetzt ist und mittels Actions ausgeführt wird:

Komponenten von GitHub Actions

Workflow	Der Workflow ist ein automatisierter Prozess eines Repositories. Workflows bestehen aus einem oder mehreren *Jobs* und können durch ein *Event* geplant oder ausgelöst werden.
Events	Ein *Event* ist eine bestimmte Aktivität, die einen Workflow auslöst
Jobs	Ein *Job* besteht aus einer Reihe von *Steps*, die auf demselben *Runner* ausgeführt werden. Ein Workflow mit mehreren *Jobs* führt diese *Jobs* parallel aus.
Steps	Ein *Step* ist eine einzelne Aufgabe, die Befehle in einem *Job* ausführen kann.
Actios	*Actions* sind eigenständige Befehle, die zu *Steps* kombiniert werden, um einen Auftrag zu erstellen. *Actions* sind der kleinste portable Baustein eines Workflows.
Runners	Ein Runner lauscht auf verfügbare *Jobs*, führt einen *Job* nach dem anderen aus und meldet den Fortschritt, die Protokolle und die Ergebnisse an GitHub zurück.

GitHub Action: Beispiel

```yaml
name: github-actions

on: [push]

jobs:
  check-bats-version:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v2
        with:
          node-version: '14'
      - run: npm install -g bats
      - run: bats -v
```

Workflow ist in **.github/workflows** gespeichert.

Shared Wiki

- **Dokumentation, Zeitplan, Tutorials, Links**

  https://github.com/jku-win-se/teaching.ws23.prse.prwiki.braeuer

Nächste Schritte

- **Jetzt**
  - ➜ Teameinteilung: 3 Personen

    1 E-Mail pro Team an johannes.braeuer@gmx.at
    Betreff: PR_SE2023 Team
    Inhalt: Name, Matr.Nr, E-Mail, **GitHub User, Clockify User**

  - ➜ Themenverteilung für Workshop

- **Bis SO 18:00 ist GitHub Repository verfügbar**

- **Bis nächste Woche (13.10.2023)**
  - ➜ Mit den Requirements vertraut machen und Fragen für den Workshop vorbereiten

  - ➜ Je nach Themenwahl mit Git, GitHub & Clockify, Maven & GitHub Actions, UML-Tools und UI Tools vertraut machen und **README** in GitHub Repository erstellen
    - Präsentation des README
    - Wissensbasis für die anderen Teams (Shared Wiki)

Themen (1)

- **Topic-1: Git (mit GitHub)**
  - ➔ Git Funktionen und Markdown
  - ➔ Git in Eclipse

- **Topic-2: GitHub & Clockify**
  - ➔ Issues, Tasks, Sprints
  - ➔ Verantwortlichkeiten, Zeiterfassung mittels Clockify

- **Topic-3: Maven & GitHub Actions**
  - ➔ Maven Projekt in Eclipse
  - ➔ Projekt mit GitHub Actions bauen

Themen (2)

- **Topic-4: UML Tools / Editoren**
  - ➔ Open Source oder Demo Version bis Februar
  - ➔ Mindestens 3 Tools (bevorzugt SaaS Lösungen)

- **Topic-5: UI Prototyping + Tools**
  - ➔ Open Source oder Demo Version bis Februar
  - ➔ Mindestens 3 Tools (bevorzugt SaaS Lösungen)


JOHANNES KEPLER
UNIVERSITY LINZ