LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Softwareentwicklungspraktikum für Fortgeschrittene

Einführungsveranstaltung

Prof. Dr. Martin Wirsing, Prof. Dr. Rudolf Haggenmüller,
Andreas Schroeder, Philip Mayer

Outline

I. **Was wir bieten**
   - Produkt
   - Prozess
   - Technologie

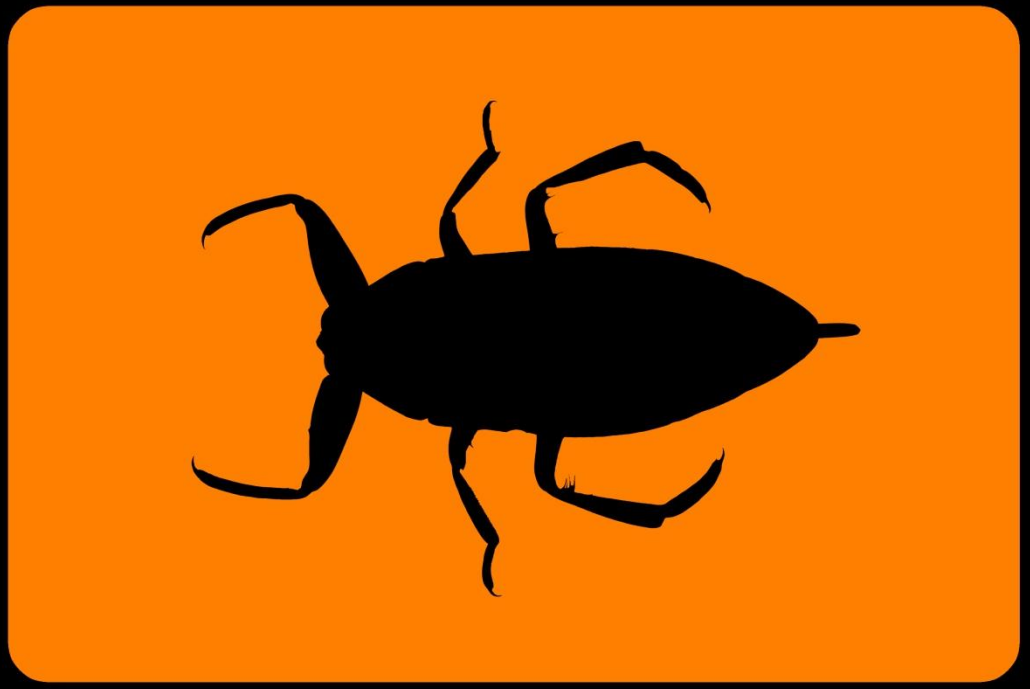
II. **Was wir erwarten**
   - Fundierte Programmierkenntnisse
   - Eigenverantwortung
   - Zeit

III. **Ablauf des Praktikums**
   - Zeitplanung
   - Benotung
   - Test

Teil I. Was wir bieten

Produkt

- **Ziele**
  - Modernes Produkt
  - Spaß
  - prinzipielle Kommerzialisierbarkeit ☺
- Das Produkt: „The Bug Is A Lie“
  - Ein Kartenspiel, welches sich in satirischer Weise der Softwareentwicklung nähert
  - Eigenentwicklung des Lehrstuhls
  - Umsetzung als **Multiplayer-(„Roleplaying“)-Onlinespiel**



The bug is a lie
The bug is a lie
The bug is a lie
The bug is a lie

The Bug Is A Lie (I)

- Spielidee:
  - Satirische Abbildung eines Software-Entwicklungsprojekts in der Industrie
- Spielablauf
  - Spieler schieben sich gegenseitig **bug reports** zu, die mittels **solution** bearbeitet oder mit einer **lame excuse** abgelehnt werden müssen
  - Spieler besitzen eine gewisse Menge **mental health**; dieses verringert sich bei nicht bearbeiteten **bug reports**. Bei **mental health == 0** wird der Spieler gefeuert

The Bug Is A Lie (II)

- Jeder der 4-7 Spieler erhält eine Rolle
  - **Manager**, **honest developer**, **evil code monkey**, und **consultant**
- Rollen (bis auf **manager**) sind am Anfang unbekannt
- Ziel des Spiels
  - Manager + Honest Developers: Evil Code Monkey und Consultant feuern
  - Evil Code Monkys: Manager feuern
  - Consultant: Firma übernehmen (=> alle anderen feuern)
- Zusätzliche **Persönlichkeitskarten** geben Sonderfähigkeiten
- Verschiedene Zusatzkarten wie Prestige, Fortbildungen etc. fügen zusätzliche Aspekte hinzu

Beispielkarten

Umsetzung

- Detaillierte Regeln werden am 26.4. geliefert

- **Umsetzung** erfolgt
  - ...als Onlinespiel
    - Lobby, Spieltisch, Chat, ggf. Statistiken
  - Implementierung eines zentralen Servers
    - Handelt user-login, game starting/stopping, game play etc.
  - Implementierung von Clients
    - UI für die Spieler
    - Kommunikation mit Server

Prozess

- **Ziele**
  - Die Idee ist, den Sinn eines Software-Entwicklungsprozesses zu erkennen
  - Der Prozess ist daher leichtgewichtig und einfach zu erlernen (no overhead)
  - Erlenen von Prozess-Techniken (Story-Planung, Tests, Bugtracking, ...)
- Der eingesetzte Prozess ist ein
  agiler Softwareentwicklungsprozess
  - Bekannte Vertreter: XP, SCRUM
  - Hier: An SCRUM angelehnter vereinfachter Prozess
  - Wird am 19.4. einer Vorlesungsstunde vorgestellt; Buch dazu ist (sehr!) empfohlen

Aufbau des Prozesses

- Handling von Funktionalität
  - Umzusetzende Funktionalität wird in **User Stories** beschrieben
  - User stories werden **priorisiert** und **geschätzt**
  - User Stories werden in **Tasks** aufgeteilt, zugewiesen und abgearbeitet
- Prozessablauf
  - Der Prozessablauf ist in **Iterationen** (alle 4 Wochen) und **Releases** (alle 3 Monate) aufgeteilt
  - Jede Iteration und jedes Release produziert einen **voll lauffähigen Zwischenstand** (wenn auch natürlich noch nicht mit kompletter Funktionalität versehen)

Prinzipien des Prozesses

- **(Selbst-)Überwachung**
  - Ein sog. **Burndown-Chart** sorgt für die lückenlose Nachverfolgung von Funktionalität und Zeitaufwand
  - Anpassung von Schätzungen gemäß tatsächlich benötigter Zeit
- **Teamwork**
  - Das Team managt sich im Prozess selbst
  - Offenheit und Ehrlichkeit prägen die Kommunikation
  - Jedes Teammmitglied erhält zudem Sonderaufgaben (wie z.B. Überwachung der Testabdeckung)

Techniken

- **Testbasierte Entwicklung**. Tests sind einer der Schlüssel zur Validierung von Funktionalität
  - Test-First oder Code-And-Test
  - Unit-Tests **müssen** entwickelt und gepflegt werden
- Nutzung von **Source Code Repositories**, **Bugtracking-Systemen**, und **Planungs-Wikis**
  - Shared Code Ownership im Repository
  - Nutzung eines Bugtracking-Systems
  - Nutzung eines Wikis zur Projektplanung inklusive **regelmäßiger** Updates des aktuellen Zustands

Technologie

- **Ziele**
  - Einsatz **produktiver** Werkzeuge (IDEs, SVN, ...)
  - Einsatz **moderner** Technologien (Mina, JUnit, ...)
- Im Speziellen:
  - **Eclipse** als IDE und Build-System
  - **Technologien:**
    - JDK 6 (Java Concurrency Framework, Swing, ...)
    - Mina (non-blocking I/O)
    - JUnit, EasyMock
  - **Verwaltungstools**
    - SVN, (Bug-)Trac, Wiki zur Planung, Mylyn

Einführung in verwendete Techniken

- Einführung in Technologien
  - Die Technologien werden am 26. April in einer eigenen Vorlesungsstunde vorgestellt
- Bootstrapping
  - Ein Code-Gerüst für Client und Server wird bereitgestellt, eine Woche Zeit für Einarbeitung
  - SVN und Trac werden zur Verfügung gestellt.

Teil II. Was wir erwarten

- Fundierte Programmierkenntnisse in Java
  - Dies ist kein Java-Kurs!
  - Wir erwarten „Java native speakers“
- Begründung:
  - Fokus soll auf dem Prozess liegen, nicht auf der Sprache
  - Level der Teilnehmer soll ungefähr gleich und adäquat für die Aufgabe sein
  - Teamarbeit sollte nicht (zu stark) durch Unterschiede in der Vorbildung gefährdet werden

Eigenverantwortung

- Wir erwarten **Interesse am Prozess** (und Produkt)
- Wir ewarten:
  - Befolgung der Regeln des Prozesses (nur so lernen Sie etwas)
  - Selbstmangement des Teams
  - Übernahme von **Verantwortung für den Code** (Team Code Ownership)
  - Ehrlichkeit und Offenheit: Einschätzung der eigenen Leistungsfähigkeit und Kenntnisse
  - Kommunikation mit dem Kunden (=uns!) und den Kollegen

- Wichtig: Die Betreuer sind in erster Linie **Kunde** – erst danach Berater bzw. „Management-Consultant“

Zeit

- Zeitmanagement
  - Zeitplanung: Ehrliche Schätzungen der Arbeitszeit
  - Zeiteinsatz: Nutzung nur der vorhandenen Arbeitszeit
    - Dies sind gemäß der CPs / SWS 1,5 Tage (=12 Stunden) pro Woche
    - **Wir erwarten den Einsatz dieser 12 Stunden!** (aber auch nicht mehr – das ist auch eines der Ziele des Prozesses)
  - Controlling: Tracking der eingesetzten Zeit und Korrektur eigener Schätzungen

- **Disclaimer**: Dies ist immer noch eine benotete Lehrveranstaltung – d.h. wir haben gewisse Vorstellungen, was in der gegebenen Zeit erledigt werden kann (s.u.)

Teil III. Ablauf des Praktikums

Zeitplanung

- Zwei Wochen Bootstrapping
  - 19. April – Vorlesung Prozess. Lesen Sie das Buch und machen Sie sich mit dem Prozess vertraut.
  - 26. April – Vorlesung Technologien. Lesen Sie den Framework-Code; programmieren Sie Spikes und testen Sie Ihre Kenntnisse
- Eigentliches SWEP
  - Drei Iterationen á 4 Wochen (Start: 3. Mai)
  - Jeweils erster Montag: Initialisierung
  - Jeweils letzter Freitag: Iterationsabschluss
  - Dazwischen jeden Montag Projektbesprechung
- Ende am Freitag, dem 23. Juli
  - ...mit dem ersten (und letzten) Release

Anwesenheits-pflicht!

Bewertungskriterien

1. Umsetzung des Prozesses

- Sinnvolle Schätzungen und Follow-Up-Korrekturen
- Stets aktuelle Projekt-Planung (im Wiki) der Stories, Iterationen, und Task Assignments
- Nutzung des Repositories und des Bugtrackers
- Korrekter Iterationsabschluss

2. Entwicklung des Produkts

- Umsetzung der Funktionalität
- Testabdeckung
- Dokumentation des Codes
- Design Principles (DRY, SRP) etc.

Test

- Ziele:
  1) Wo stehen Sie als Software-Engineer (bezüglich Java, Nutzung von Bibliotheken, IDEs, ...)
     - Ziel: **Auswahl**
  2) Wo steht das Team insgesamt
     - Ziel: **Feedback für uns**
- Wir erwarten keine perfekte, sondern eine **gut designte** Lösung
- 60 Minuten
- Notification: Freitag per E-Mail (Uniworx-Anmeldung erforderlich)

Zusammenfassung

The Last Slide

- Überlegen Sie sich,
  - Ob das Praktikum für Sie interessant ist
  - Ob Sie bereit sind, die Erwartungen zu erfüllen
- **15 Min Pause**
- Danach: Test