# Blockpraktikum Multimediaprogrammierung

15. September - 26. September 2008

Max Maurer

Erfahrungsbericht Extreme Programming

# Kleiner Rückblick zur Vorlesung

## The XP Team

- Small group, at most 12 persons
- Group should fit into a single room
  - Ideally the group is co-located for the whole working time
- Special member roles:
  - Representatives of the customer ("on-site customer")
  - *Tester:* Helps the customer to translate his stories into functional tests
  - *Coach:* Helps in keeping the XP discipline
  - *Tracker*: Continuously measures progress of the team and publishes it
  - *Consultant:* External provider of specific knowledge, active on the team only for a short time
  - *Big Boss:* Encourages the team

Ludwig-Maximilians-Universität München | Prof. Hußmann | Praktikum Multimedia-Programmierung – 7

Prof. Heinrich Hußmann
**"Agile Development for Multimedia Projects"**
2008

# Kleiner Rückblick zur Vorlesung

## Four Values of XP

- Communication
  - XP aims to keep the right communication flowing
  - XP defines practices that require communication
- Simplicity
  - *What is the simplest thing that could possibly work?*
  - Do a simple thing today and pay a little more tomorrow to change it if needed
  - Do not do a complicated thing today that may never be used anyway
- Feedback
  - *Don't ask me, ask the system! Have you written a test case for that yet?*
  - Minute-to minute scale feedback:
    - » Feedback about system state due to extensive automated testing
  - Week-to-month scale feedback:
    - » Customers get updates about the growth of the system
- Courage
  - Fix flaws, don't circumvent them. Throw code away if it is unstable.

Ludwig-Maximilians-Universität München | Prof. Hußmann | Praktikum Multimedia-Programmierung – 8

Prof. Heinrich Hußmann
**"Agile Development for Multimedia Projects"**
2008

# Kleiner Rückblick zur Vorlesung

## Elements of XP and their Applicability to Multimedia Authoring

| | |
|---|---|
| • The Planning Game | applicable directly |
| • Small releases | applicable directly |
| • Metaphor | applicable in adapted way |
| • Simple design | applicable in adapted way |
| • Testing | How to automate tests for Flash? |
| • Refactoring | applicable in adapted way |
| • Pair programming | applicable directly |
| • Collective code ownership | applicable directly |
| • Continuous integration | applicable directly |
| • 40-hour week | applicable directly |
| • On-site customer | applicable directly |
| • Coding standards | applicable / which standards? |
| • *Easily modifiable program code* | *media objects difficult to modify* |

Ludwig-Maximilians-Universität München Prof. Hußmann Praktikum Multimedia-Programmierung – 26

Prof. Heinrich Hußmann
**"Agile Development for Multimedia Projects"**
2008

# Zeiträume

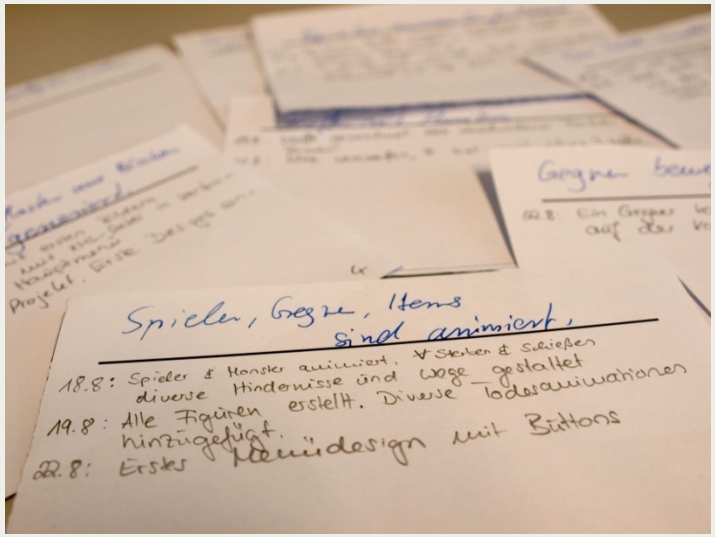
- Klassisches XP kennt folgende Zeiträume
  - Releases: Ein bis sechs Monate
  - Iteration: Eine bis drei Wochen
- Im Blockpraktikum:
  - Nur sechs Tage effektive Entwicklungszeit
  - Reduzierung der Release-Zeit auf sechs Tage
  - Iteration bereits nach drei Tagen

# 1: The Planning Game

- Klassisch:
  - Teilnehmer am Planning Game
    - Kunden (Business people) unterbreiten Vorschläge
    - Programmierer (Technical people) schätzen die Kosten der Vorschläge
- Im Blockpraktikum:
  - Nicht immer eine strikte Trennung möglich
  - Oft auch Vorschläge aus den Reihen der Entwickler

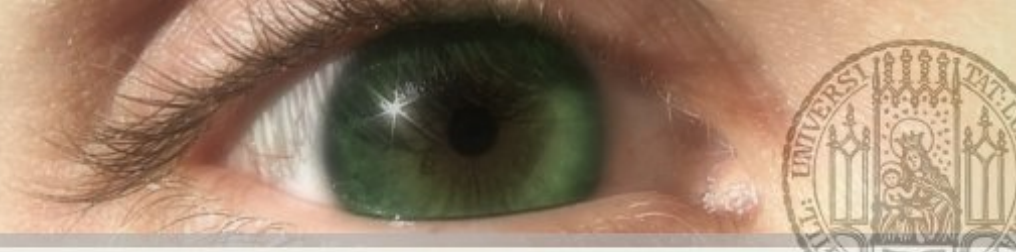
# User Stories

- Klassisch:
  - In Alltagssprache formulierte Software-Anforderung
  - Dokumentiert die im Planning Game entstandenen Konventionen
- Im Blockpraktikum:
  - Spezifikation von verschiedenen User-Stories
  - Prüfung des Entwickelten anhand der gemachten User Stories in der Interation
  - Entwicklung hauptsächlich mit Hilfe des Metaphers; User Stories im Hinterkopf



# 2: Small Releases

- Klassisch:
  - Kurze in sich geschlossene Releases
    - Alle ein bis zwei Monate
- Im Blockpraktikum:
  - Keine festen Releases, aber zum Tagesende meist ausführbarer teilfunktionaler Cod

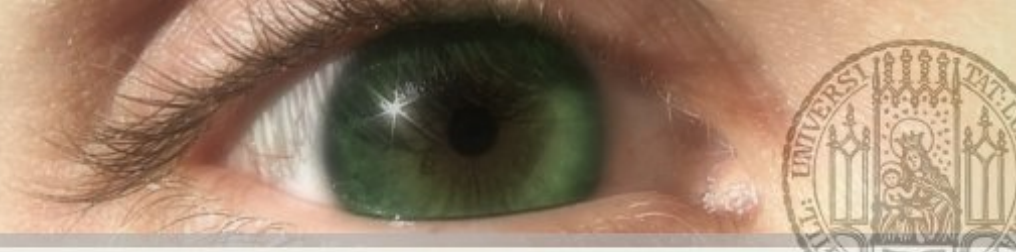


# 3: Metaphor

- Klassisch:
  - Methaper beschreibt das Gesamtsystem
    - Begriffe der realen Welt werden verbunden (Spieler, Level)
- Im Blockpraktikum:
  - Schon bei einem kleinen Spiel sind die Beziehungen zwischen den Begriffen nicht logisch erkennbar
  - Zumindest ein UML Diagramm das die großen Beziehungen modelliert ist sinnvoll

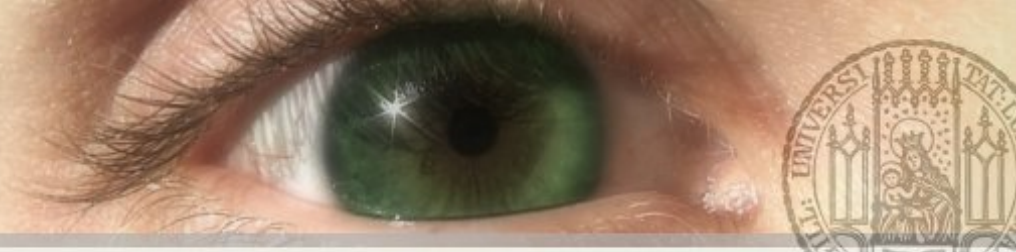
# 4: Simple Design

- Klassisch:
  - Das einfachste System ist das richtige, sofern es:
    - Alle Tests erfüllt
    - Keine doppelten Algorithmen enthält
    - Alle als wichtig angesehenen Anforderungen erfüllt
    - Mit den wenigsten Klassen und Methoden auskommt
- Im Blockpraktikum:
  - Start mit einer leeren Datei
  - Hinzufügen von Methoden und Klassen erst, wenn die Notwendigkeit entstand
  - Probleme entstehen aber bei der Müllbeseitigung. Durch verschiedene Code Autoren, weiß niemand ob eine Methode vielleicht nicht mehr gebraucht wird.



# 5: Testing...

- Klassisch:
  - Test-first-Ansatz
  - Automatisierte Tests
  - Tests ersetzen die traditionelle Software-Spezifikation
- Im Blockpraktikum:
  - Tests vor dem Code zu entwickeln stellt große Probleme dar:
    - Ohne Spezifikation ist die Klassenstruktur unklar
    - Tests sind aber Klassenstrukturbasiert
  - In grafischer Software besitzen die meisten Methoden keine Rückgabewerte, sondern verursachen indirekt Auswirkungen auf die grafischen Objekte. Dies ist schwer testbar
  - Notwendigkeit für Tests wäre aber gerade bei solchen Methoden groß
  - In kleinen Methoden mit Rückgabewerten meist kaum Fehlerquellen
  - Zeitfaktor, Motivationfaktor

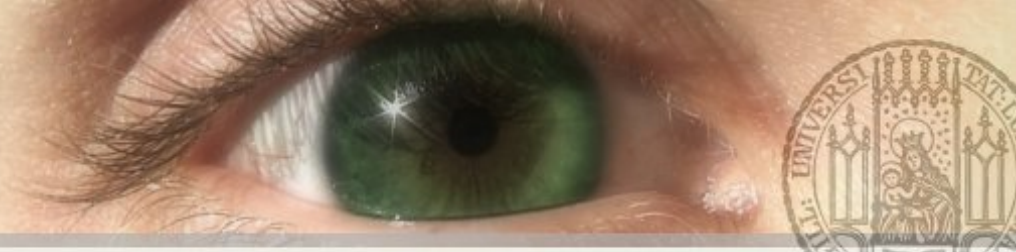


# 6: Refactoring

- Klassisch:
  - Neue Features nicht unbedingt immer an den vorhanden Code anknüpfen
  - Wenn notwendig erst umstrukturieren
  - Nach dem Umstrukturieren laufen alte Tests immer noch einwandfrei
- Im Blockpraktikum:
  - Parallele Entwicklung führt zu verschiedenem Code
  - Neue Features benötigen neue Klassen
  - Codebasis wurde stets an neu geplante Features angepasst

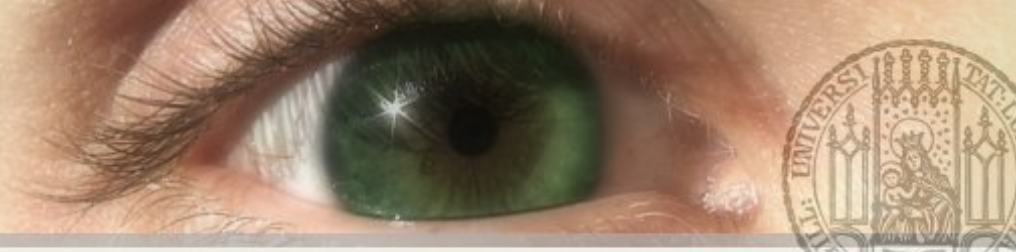
# 7: Pair programing

- Klassisch:
  - Programmierer (an Tastatur und Maus)
  - Beobachter: Kontrolliert und versucht die Verbindung zum Gesamtsystem zu beachten
- Im Blockpraktikum:
  - Je nach Team mehr oder weniger benutzt
  - Häufig dadurch Fehler vermeiden möglich
  - Zur expliziten Fehlersuche auch kurzzeitiges aufsplitten eines Teams an zwei Rechner möglich



# 8: Collective Ownership

- Klassisch:
  - Code hat an jeder Stelle verschiedene Autoren
  - Jeder darf jeden Code sehen und bearbeiten (nach Absprache)
  - Für gewisse Codestellen gibt es eventuell Experten
- Im Blockpraktikum:
  - Räumliche Nähe der Beteiligten sorgt für gute Kenntnisse des Systems
  - Schnelle Nachfrage beim Teamkollegen möglich
  - Teamgröße optimal um gute Zusammenarbeit zu gewährleisten



# 9: Continuous integration

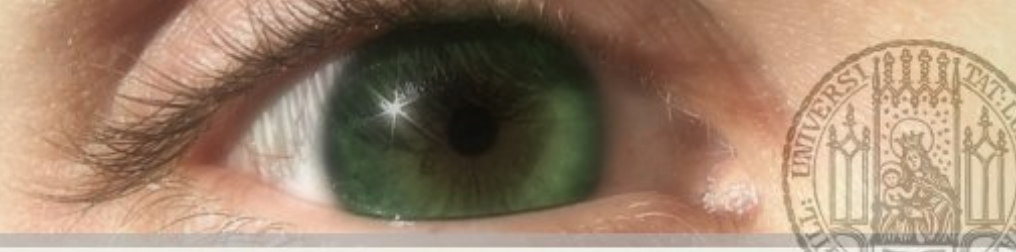
- Klassisch:
  - Es gibt immer ein aktuelle lauffähige Version
  - Integration von neuem Code wird täglich durchgeführt
  - Es gibt eine extra „integration machine“ die von allen Entwicklern genutzt wird
- Im Blockpraktikum:
  - Code Entwicklung auf verschiedenen Rechnern
  - Integration direkt ins SVN oder gemeinsam an einem Rechner
  - Integration mehrmals pro Tag

# 10: 40-Stunden-Woche

- Klassisch:
  - „sustainable pace“ gewährleistet eine dauerhaft aufrecht zu erhaltende Entwicklungsgeschwindigkeit
  - Maximale eine Woche mit Mehrarbeit ist erlaubt
- Im Blockpraktikum:
  - 40-Stunden-Woche auch im Blockpraktikum
  - Volle Konzentration ist nicht mehr als 8 Stunden am Tag möglich
  - Ausgeruht lässt sich deutlich schneller und fehlerfreier entwickeln

# 11: On-Site Customer

- Klassisch:
  - Ein Kunde oder mehrere Repräsentative des Kunden bilden einen Teil des Entwicklungsteams
  - Ständige Verfügbarkeit als Ansprechpartner
- Im Blockpraktikum:
  - On-Site-Customer nahm mehrere Rollen parallel ein
  - Häufig können Anfragen auch unbeantwortet an die Entwickler zurück gegeben werden
  - Gute Kontrollmöglichkeit, schützt vor Fehlentwicklungen



# 12: Coding Standards

- Klassisch:
  - Namenskonventionen
  - Dokumentationskonventionen
  - Code Layout Konventionen
  - Frameworks
- Im Blockpraktikum:
  - Eigenständige Coding Standards in den jeweiligen Teams (Wiki und SVN)
  - Namenskonventionen entwickeln sich
  - Durch räumliche nähe entwickeln sich Coding Standard fast automatisch
  - Nicht standardisierte Informationen können vom jeweiligen Experten erfragt werden