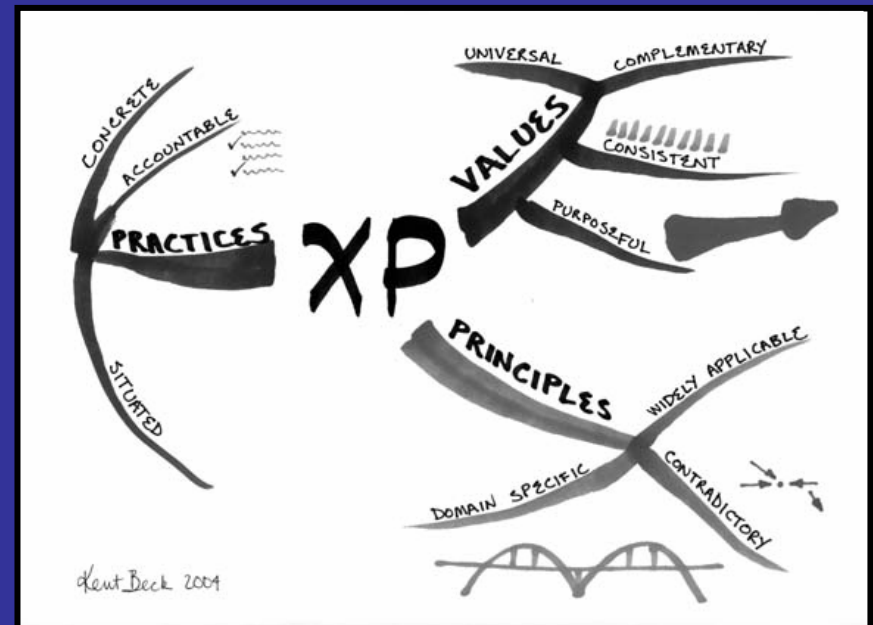


eXtreme Programming (XP)

Softwaretechnik SS2005

Tobias Giese
Masterstudiengang Informatik
HS-Harz

29.06.2005




◆ Agenda

- Allgemeines
- Vorgehensmodell
- Kommunikation und Arbeitsphilosophie
- Entwicklungsphasen / Extreme Rules
 - Planung
 - Entwurf
 - Implementierung
 - Testen
- Management von XP-Teams

Extreme Programming?

◆ Allgemeines

- bekanntes **agiles** Vorgehensmodell
- erste Publizierung: Kent Beck
- basiert auf **Erfahrungen** und **Best-Practises**
- **extreme** = Praktiken in extremen Umfang 
- fordert strenge **Disziplin-** und **Prinzipientreue**
- **Anpassung** des Entwicklungsprozesses an **örtliche** Gegebenheiten

Extreme Programming = \sum Extreme Rules

Extreme Rules

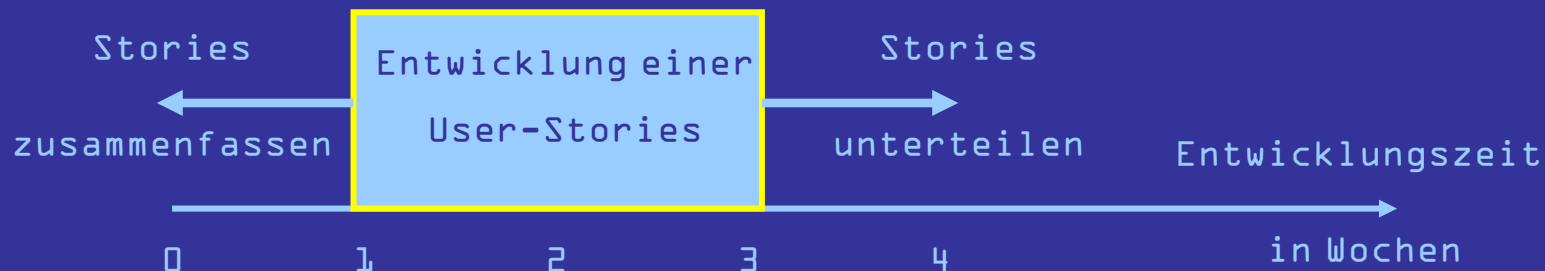
◆ Vorgehensmodell

- **iteratives** und **evolutionäres** Vorgehen (kleine Releases)
- permanentes **Testen**
- **2-12** Entwickler
- **knapp**e Analyse- und Designphase (Anforderungen u. Architekturentwurf)
- flexibles **Anforderungsmanagement**
- **Refactoring** (Designverbesserung)
- **kleine Releases** = voll lauffähige Softwaresysteme (Abnahme)
→ frühzeitiges **Feedback** → höhere **Agilität** (Design + Features)
- periodische **Release Planning Meetings** für **Synchronisierung**

Planning → Designing → Coding → Testing

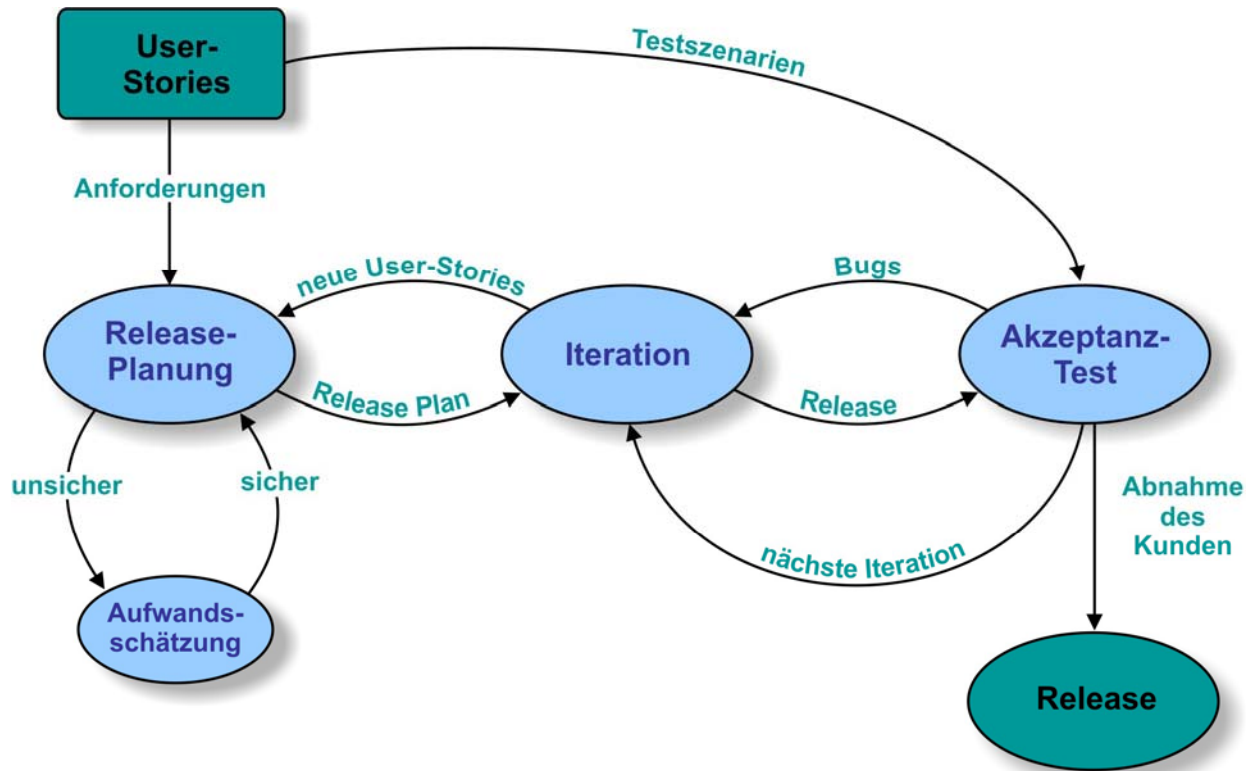
◆ User Stories – Fakten

- **Anforderungsdokument** (~ Use-Cases)
 - Unterschiede: Detailgrad → **generischer Ansatz**
 - enthalten **reine** Anforderungen → (Technologien || Algorithmen)
- 2 Phasen während der Entwicklung
 - 1. in der **Analysephase** (grob) : Abschätzung, Planung
 - 2. in der **Implementierungsphase** (fein) : Kontakt mit Kunden
- vom **Kunden** angefertigt (Lastenheft)
- dienen als **Testszenario** für den **Akzeptanztest** (Abnahmekriterien)
- Ziel in XP: **80** User-Stories (+/-20) → Erstellung: **Release Plan**



Planning → Designing → Coding → Testing

User Stories / Release Planung - Überblick



Planning → Designing → Coding → Testing

◆ Kommunikation und Arbeitsphilosophie

- enge Zusammenarbeit der Teammitglieder (**fachlich** wie **physikalisch**)
- **Pair Programming** ↔ Trainings Wissensweitergabe 😞
 - verhindert Wissensinseln
 - **gleichverteiltes** Wissen → **gleichverteilte** Arbeitsbelastung (flexibles Load-Balancing) 😊
- Tägliche **Stand Up Meetings** (nur ein Meeting mit allen Entwicklern, statt vieler kleiner, meist vor Iterationsplanung) im Stehen → agil
- **keine** Überstunden! (senkt Geist & Motivation)
- **Optimierungen** erst am Ende
 - “Make it **work**, make it **right**, then make it **fast**.” (Kent Beck)
- **Retrospektiven** (was war nicht so gut → Verbesserungen)

Whiteboardeinsatz

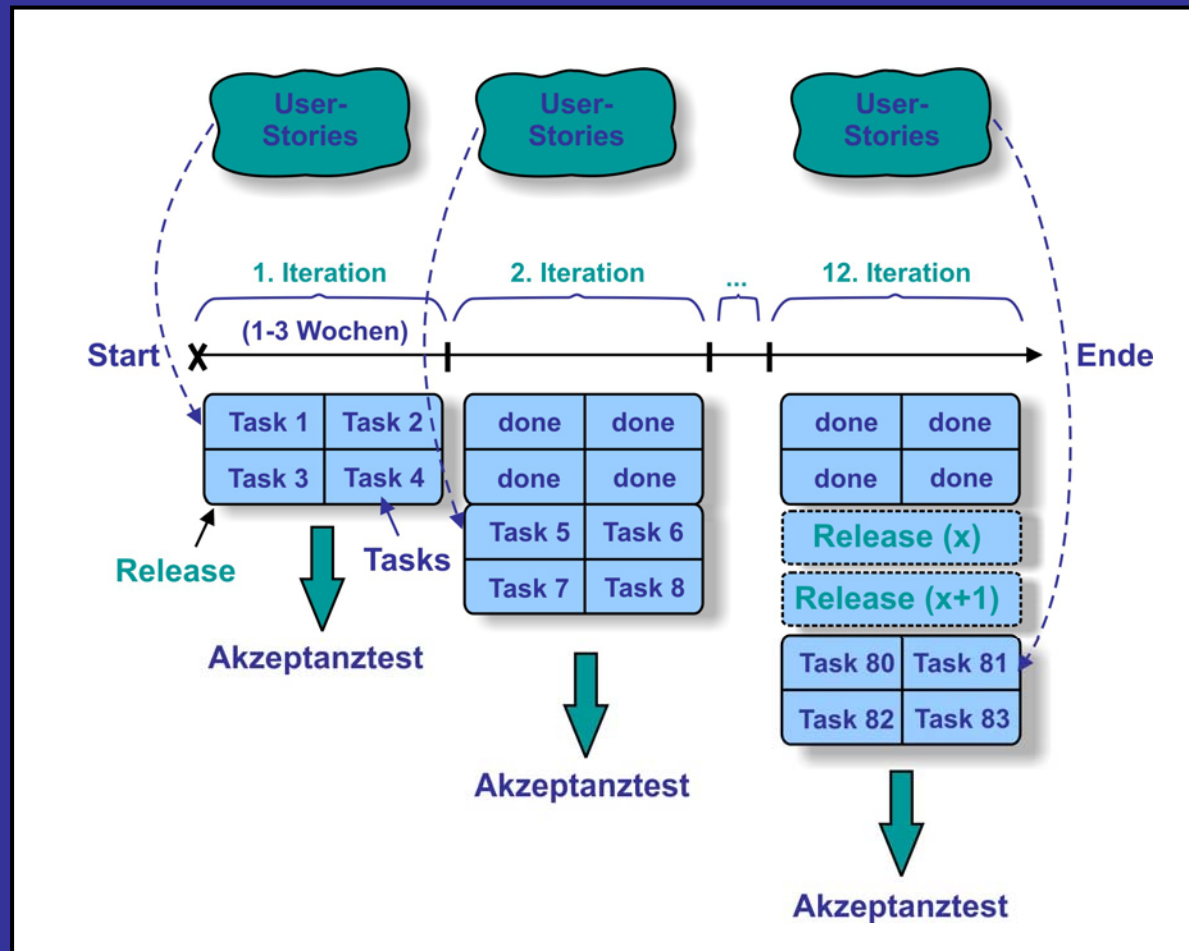
Planning → Designing → Coding → Testing

◆ Iterative Entwicklung - Fakten

- 12 Iteration a 1-3 Wochen (**konstante** Länge) → Kern von XP
 - unterstützt das **Messen** der Projektfortschrittes
- kick-off: **Iterations-Planungs-Meeting** (Just-in-Time) → Agilität
- Entwicklungsstrategie: **straight forward** (most-important-story)
- Iterations **Deadlines** haben **hohe** Priorität
- Wenn **Regeln** gebrochen → **Iterations Planungs Meeting**
- **Kundenanforderungen** haben Vorrang (Kunde priorisiert User-Stories) → ausgewählt aus **Release Plan**
- **User-Stories** (Kunde) ↔ **Tasks** (Entwickler) = Index Karten, die als detaillierter Plan für die Iteration dienen
- **Abschätzung** der Dauer durch den Entwickler der den Task bearb.

Planning → Designing → Coding → Testing

◆ Iterative Entwicklung - Überblick



Planning → **Designing** → Coding → Testing

◆ Design

- oberste Maxime: **Einfachheit**
- *“Keep things as **simple** as possible as long as possible by **never adding functionality** before it is **scheduled**.”* (Kent Beck, Extreme Programming Explained)
- **einheitlicher** Coding Style (Klassen-/Methodenbezeichner)
- **Class**
Responsibilities
Collaboration
- = einheitliche Sprache des Teams
 - Cards** für den **Entwurf**
 - Vorteil: alle Teams können den Entwurf mitgestalten
- ständiges **Refactoring** (Entfernen von **Redundanzen** / **Funktionalitäten**, **Designverbesserung**) → **Straight-forward** = refactor mercilessly

Planning → Designing → **Coding** → Testing

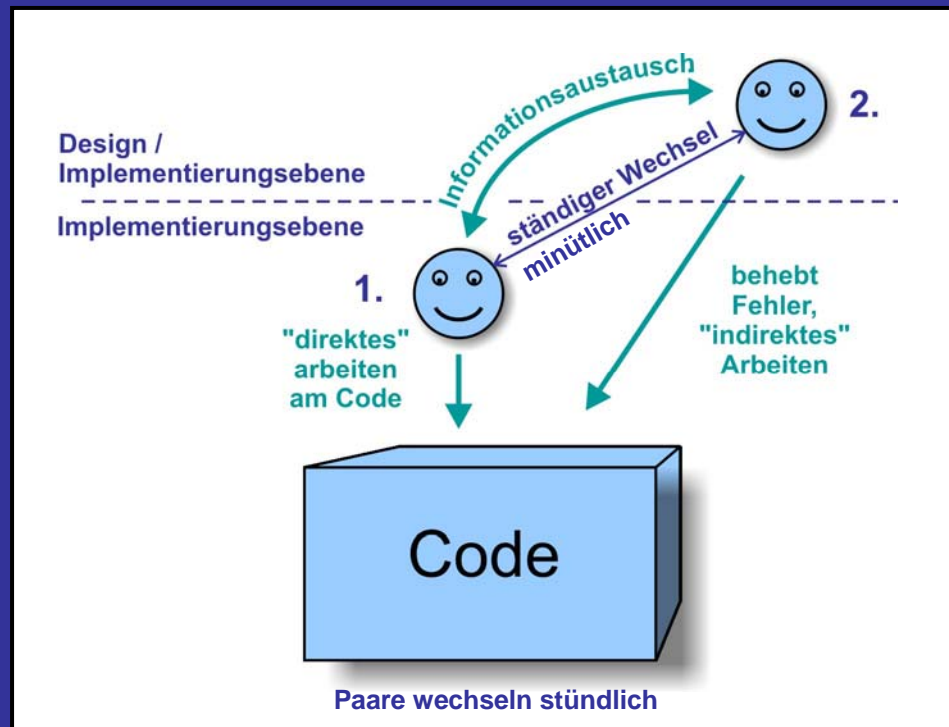
◆ Coding

- laufender **Kundenkontakt** (Teil von XP)
- **Kunde** verantwortlich für **User-Stories**
- **Verfeinerung** der User-Stories → **Programmer Tasks**
- 1.Schritt: Programmierung der **Unit Tests**
 - Fixierung der Anforderungen im Test → keine Mißverständnisse (ausführbare Spezifikation)
 - sofortiges Feedback (wann bin ich fertig?)
 - besseres, einfacheres Design (Test-Driven-Development)
- 2. Schritt: Programmierung des **Codes**
- **Sequentielle Integration** in das Code-Repository (single threaded)
 - nur jeweils ein Developer-Pair darf Code einchecken (lock)
 - alle paar Stunden → jeder Developer arbeitet an **akt. Version**

Planning → Designing → **Coding** → Testing

◆ Pair Programming

- **paarweises** Arbeiten an Design und Code



- **frühe** Fehlererkennung
- **höherer** Produktivität
- **höherer** Spaßfaktor
- starker **Informationsaustausch**



- **doppelter** Personaleinsatz
- **Zwang** zur Kommunikation
- **Absolute** Teamfähigkeit ist Voraussetzung

Planning → Designing → Coding → **Testing**

◆ Testing

- Kunde entwirft **Testdaten**
- Eckstein von XP: automatische **Unit Tests**
 - Testen **aller** Klassen → **Ziel → jederzeit muss eine lauffähige Version vorliegen**
 - Tests **first**
 - zusammen mit Code ins **Repository**
 - **zeitaufwendig** (spart aber aufwendiges Bugfixing am Ende)
- beweisen die volle **Funktionsfähigkeit** des Releases
→ **Collective Code Ownership** wird ermöglicht
- unterstützen **Refactoring**
 - Test muss **ohne** Fehler durchlaufen
 - **Regressions-** und **Validierungstests**
- **Synchronisation** Code ↔ Tests (akt. Version werden benötigt)

Planning → Designing → Coding → **Testing**

◆ Testing

- **Akzeptanztest** für jede Iteration (aus User-Stories)
 - beweist das die vom Kunden gewünschten **Anforderungen** (diese User-Story) **funktionieren** und er das Softwareprodukt **akzeptiert**
- **simuliert** den Einsatz beim Kunden
 - findet in der Firma statt
 - automatisiert für viele Tests während der Entwicklung
 - **Ergebnisse** gehen an den **Kunden** → Review & priorisieren von Bugs und an die **Developer**
- **erfolgreicher** Akzeptanztest == User-Story **komplett** implementiert

◆ Management von XP-Teams

- Akzeptierte Verantwortung
 - **keine Vorschriften** seitens der Manager an die XP-Teams
- Permanente IST-Analyse durch **Metriken**
 - zeigen **Fortschritt** des Teams
 - oder die noch zu lösenden **Probleme**
 - erkennen wo **Hilfe** notwendig ist
- Konstante Entwicklungsgeschwindigkeit
 - **keine** Überstunden
 - bei Terminproblemen Team den Rücken freihalten

Planning → Designing → Coding → **Testing**

◆ Extreme Rules in a nutshell (XP-Regeln)

1. iteratives und evolutionäres Vorgehen
2. permanentes Testen (Code + Test)
3. Just-in-Time Entwicklung (nur das aktuelle, nur das nötigste)
4. Refactoring → Loslassen von alten Ideen/Entwürfen
5. Retrospektiven → Verbesserung des Entwicklungsprozesses
6. kleine Releases → frühes Feedback → flexible Anforderungen
7. Kommunikation & Synchronisierung
8. Pair Programming
9. Collective Code Ownership → einheitliche Codebasis
10. einfaches Design
11. starke Kundenorientierung (User-Stories / Testszenarien)

Fragen?



Antworten!

Vielen Dank für ihre Aufmerksamkeit!

Tobias Giese

29.06.2005

eXtreme Programming (XP)

Kontakt:

tobias.giese@gmx.de

Unterlagen:

<http://www.tobias-giese.de>

