

Kapitel 13.

Agile Softwareentwicklung und Extreme Programming (XP)

Stand: 1.2.2011

Was sind „Agile Methodologien“?

- Eine **Methodologie** ist eine bestimmte Art und Weise den Softwareentwicklungsprozess zu organisieren. Sie legt fest:
 - ◆ Was wir tun
 - ◆ Wann wir es tun
 - ◆ Welche Werkzeuge wir benutzen
 - ◆ Wie wir den Prozess planen
 - ◆ Wie wir den Prozess kontrollieren
 - ◆ ...
- Eine **Agile Methodologie** ist eine Methodologie die versucht die Entwicklung zu beschleunigen durch:
 - ◆ **Zusammenarbeit mit dem Kunden** ↔ Anstelle von Vertragsverhandlungen
 - ◆ **Auf Änderungen reagieren** ↔ Anstatt einem Plan zu folgen
 - ◆ **Funktionierende Software** ↔ Anstelle von vollständiger Dokumentation
- Beispiel: „**Extreme Programming**“

Was ist "eXtreme" Programming?

"Extreme Programming setzt bekannte Prinzipien und Praktiken extrem konsequent um."

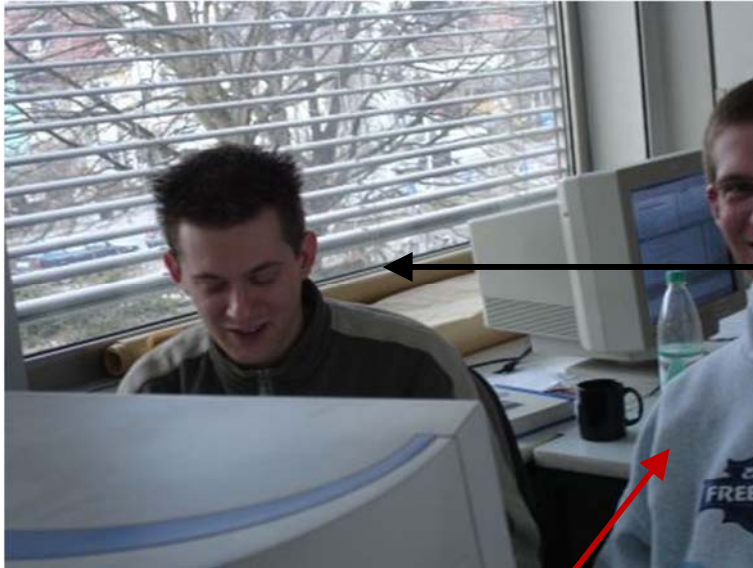
Bekannte Prinzipien und Praktiken

- Feedback ist gut
- Code reviews sind gut

... extrem konsequent umgesetzt

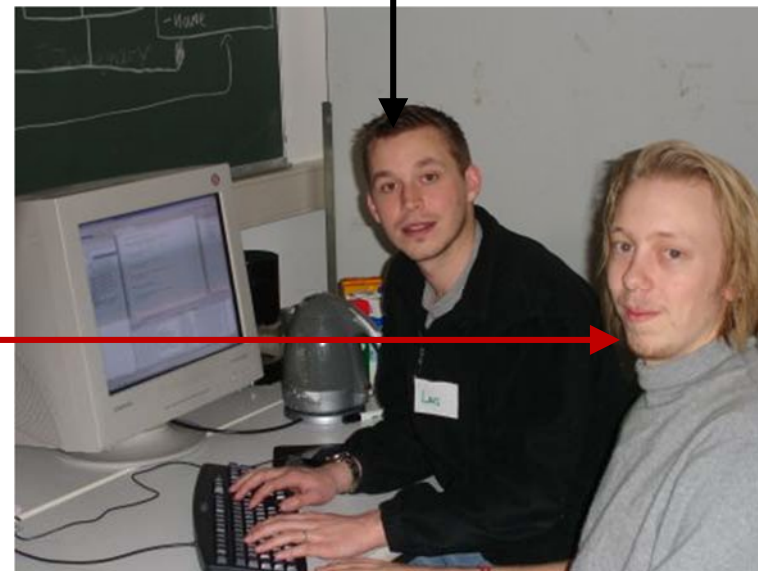
- ➔ Kunde ist ein Teil des Teams
 - ◆ permanentes Kundenfeedback
- ➔ Pair Programming
 - ◆ permanente Code Reviews!

Pair Programming: 2 Partner, 1 Computer



1 Partner programmiert.

**Der 2. Partner überprüft den Code.
... oder plant voraus.**



Was ist "eXtreme" Programming?

"Extreme Programming setzt **bekannte Prinzipien** und Praktiken **extrem konsequent** um."

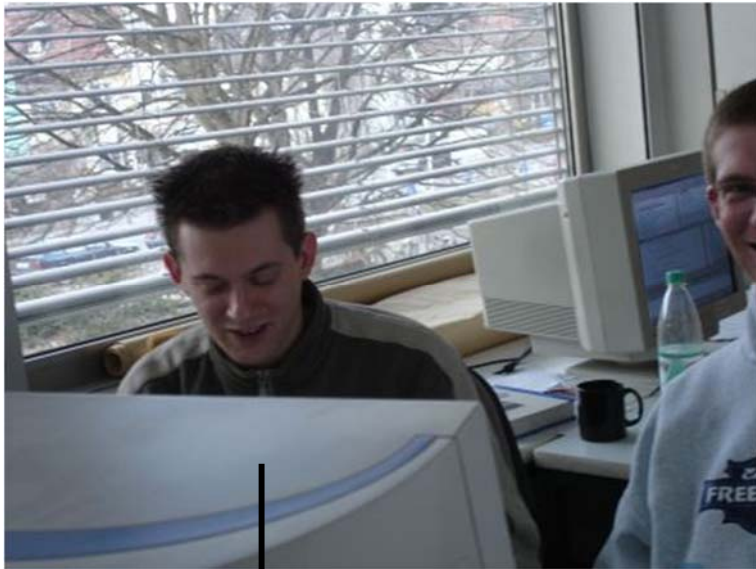
Bekannte Prinzipien und Praktiken

- Feedback ist gut
- Code reviews sind gut
- Testen ist gut
- Integrationstests sind gut

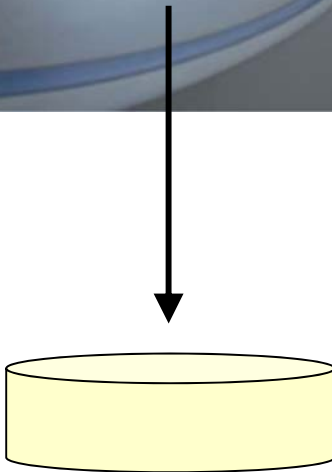
... extrem konsequent umgesetzt

- ➔ Kunde ist ein Teil des Teams
 - ◆ permanentes Kundenfeedback
- ➔ Pair Programming
 - ◆ permanente Code Reviews!
- ➔ Permanentes testen!
 - ◆ Programmierer: Unit tests
 - ◆ Kunde: Funktionstests
- ➔ Kontinuierliche Integration
 - ◆ So oft wie möglich

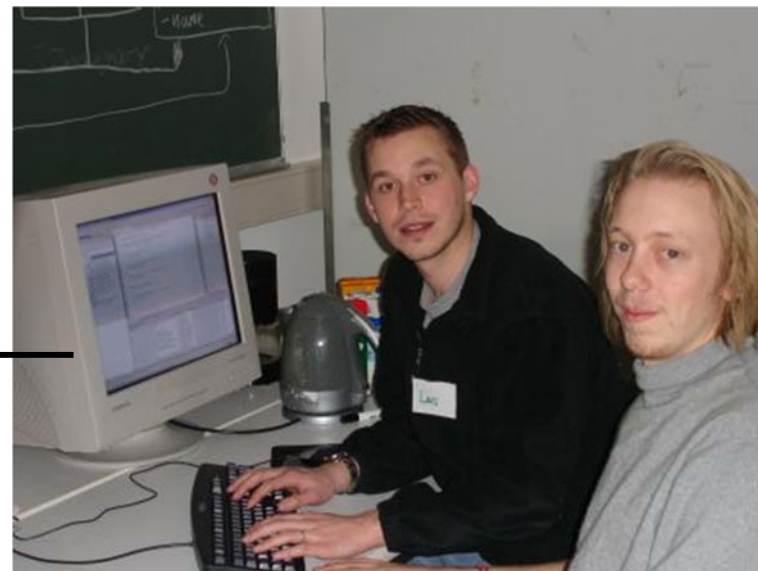
Continuous Integration: Integriere so oft wie möglich!



Integriere nur wenn
alle
Tests erfolgreich waren!



Gemeinsames Repository (CVS)



Was ist "eXtreme" Programming?

"Extreme Programming setzt bekannte Prinzipien und Praktiken extrem konsequent um."

Bekannte Prinzipien und Praktiken

- Feedback ist gut
- Code reviews sind gut
- Testen ist gut
- Integrationstests sind gut
- Design ist wichtig
- Einfachheit ist gut
- Kurze Iterationen sind gut

... extrem konsequent umgesetzt

- ➔ Kunde ist ein Teil des Teams
 - ◆ permanentes Kundenfeedback
- ➔ Pair Programming
 - ◆ permanente Code Reviews!
- ➔ Permanentes testen!
 - ◆ Programmierer: Unit tests
 - ◆ Kunde: Funktionstests
- ➔ Kontinuierliche Integration
 - ◆ So oft wie möglich
- ➔ Refactoring
 - ◆ permanentes (Re)Design
- ➔ ziehe einfache Lösungen vor
 - ◆ Refaktoriere später, wenn nötig
- ➔ „Planning Game“
 - ◆ Sehr kurze Iterationen

Planning Game: Schätze Kosten und Risiken → Plane ein Release

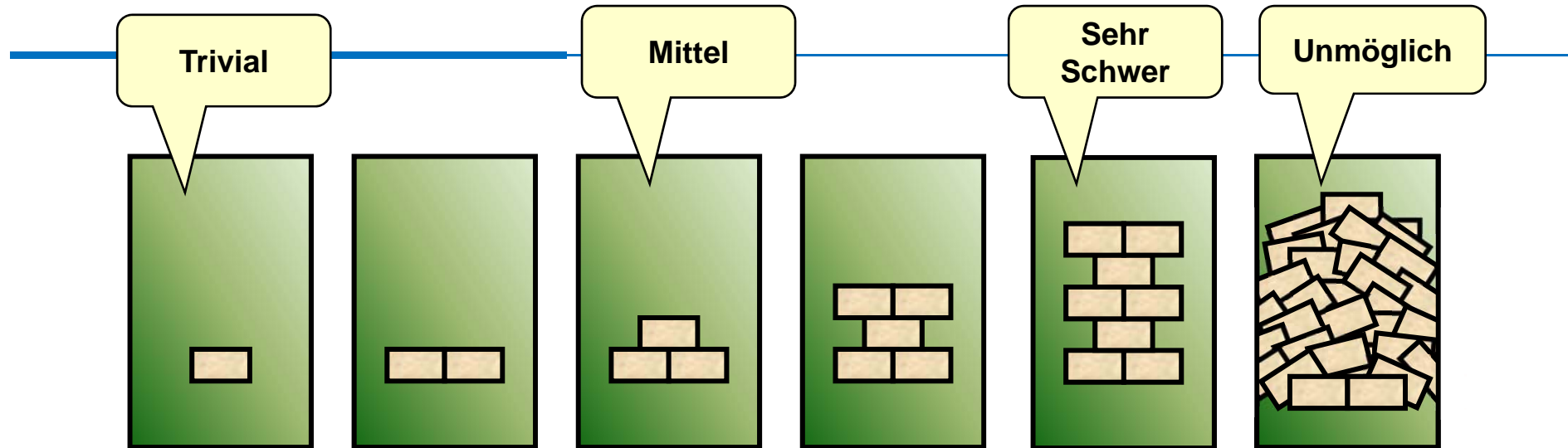
Schritte des „Planning Game“

- Identifiziere „Stories“ (= Funktionalität die der Kunde benötigt)
 - Bestimme eine Priorität für jede Story
- Kunde**

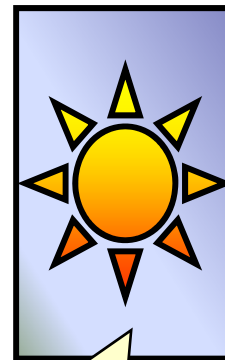
- Schätze die Kosten für jede Story
 - Schätze die Risiken für jede Story
- Programmierer**

- Lege die zu implementierende Funktionalität fest
 - ◆ 1 Iteration = 5 Arbeitstage
 - ◆ Wähle die Stories aus, die in der nächsten Iteration realisierbar sind
 - ◆ Stelle die anderen zurück
- Gemeinsam**

Planning Game: Schätze Kosten und Risiken → Plane ein Release



Risiko, daß der benötigte Aufwand viel **höher** sein könnte.



Chance, daß der wirkliche Aufwand viel **niedriger** sein könnte.

Das XP-Game (basierend auf <http://www.xp.be/xpgame.html>)

**Story
Beschreibungen**

“Planning Poker” ...




Story Cards



Issue Tracking System „Jira“: Story

[http://roots.iai.uni-bonn.de/jira32/secure/IssueNavigator.jspa?mode=hide&requestId=10090](#)
[Go](#)



Research on Object-Oriented Technologies and Systems

User: Daniel Speicher
[History](#)
[Filters](#)
[Profile](#)
[Log Out](#)

[HOME](#)
[BROWSE PROJECT](#)
[FIND ISSUES](#)
[CREATE NEW ISSUE](#)
[ADMINISTRATION](#)

Filter: [View](#) [Edit](#) [New](#) [Manage](#)

Name:
Prompter stories

Description:
All stories of the project prompter.

Summary

☒ **Project:** [Prompter](#)
☒ **Issue Type:** Story
☒ **Sorted by:** Fix Version/s ascending, then Summary ascending, then Priority ascending

Operations

☐ [Rename](#) current filter
☐ [Save as](#) new filter
☐ [Subscriptions](#) for current filter
☐ [Share](#) current filter

Issue Navigator -- Prompter stories

Displaying issues **1** to **20** of 22 matching issues.


Current View:
[Browser](#) | [Printable](#) | [XML](#) | [Full Content](#) | [Excel](#) ([All fields](#) | [Current fields](#)) ?

☐ [Bulk Change](#)
☐ [Configure](#) yo

1 | 2 | [Next >>](#)

T	Key	Summary	Assignee	Pr	Status	Res	Created	Updated	Fix Version/s	Orig
	PR-3	S00 Prepare developement infrastructure	Daniel Speicher	↑	Closed	Fixed	02/Nov/05	06/Dec/05	0.2M1	
	PR-1	S01 Easy Checkout	Daniel Speicher	↑	Closed	Fixed	02/Nov/05	15/Jan/06	0.2M1	16 h
	PR-14	S02 SPIKE Interaction categorisation + indexing	Unassigned	↑	Closed	Fixed	02/Nov/05	05/Dec/05	0.2M1	
	PR-17	S03 Prototype executably locally on laptop	Unassigned	↑	Open	UNRESOLVED	02/Nov/05	07/Nov/05	0.2M1	
	PR-11	S04 Anwendungen ohne Kategorisierung	Unassigned	↑	Open	UNRESOLVED	02/Nov/05	06/Feb/06	0.2M1	
	PR-13	S05 Static test coverage >= 30%	Unassigned	↑	Open	UNRESOLVED	02/Nov/05	18/Nov/05	0.2M1	
	PR-53	S14 Move scenario specific data into separte projects	Unassigned	↑	Open	UNRESOLVED	09/Jan/06	13/Feb/06	0.2M1	12 h
	PR-55	S15 Create Gui for CreateData	Bernd Wahlen	↑	Closed	Fixed	09/Jan/06	10/Jan/06	0.2M1	3 ho
	PR-57	S16 Gui for manual Locaton-Change	Bernd Wahlen	↑	Closed	Fixed	09/Jan/06	13/Jan/06	0.2M1	16 h
	PR-71	S18 Remove static dependency on the indexing bundle from the mail and notes client	Unassigned	↑	Open	UNRESOLVED	06/Feb/06	14/Feb/06	0.2M1	

Issue Tracking System „Jira“: Stories und






Research on Object-Oriented Technologies and Systems

User: Daniel Speicher [History](#) | [Filters](#) | [Profile](#) | [Log Out](#)

[HOME](#) | [BROWSE PROJECT](#) | [FIND ISSUES](#) | [CREATE NEW ISSUE](#) | [ADMINISTRATION](#) | [QUICK SEARCH:](#)

Issue Details

[\[XML\]](#)

Key: [PR-1](#)
Type:  Story
Status:  Closed
Resolution: Fixed
Priority:  Major
Assignee: [Daniel Speicher](#)
Reporter: [Daniel Speicher](#)
Votes: 0 [\(View\)](#)
Watchers: 0 [\(View\)](#)

Available Workflow Actions

- ☐ [Reopen Issue](#)

Operations

- ☐ [Assign](#) this issue
- ☐ [Clone](#) this issue
- ☐ [Comment](#) on this issue
- ☐ [Delete](#) this issue
- ☐ [Link](#) this issue to another issue
- ☐ [Move](#) this issue
- ☐ [Voting:](#)
You cannot vote or change your vote on resolved issues.
- ☐ [Watching:](#)
You are not watching this issue.
[Watch it](#) to be notified of changes
- ☐ [Worklog:](#)
Worked on this issue? [Log work done](#)

Prompter

S01 Easy Checkout

Created: 02/Nov/05 02:26 PM Updated: 15/Jan/06 02:09 PM

Component/s: None
Affects Version/s: None
Fix Version/s: [0.2M1](#)

Original Estimate:	2 days	Remaining Estimate:	2 days	Time Spent:	Unknown
---------------------------	--------	----------------------------	--------	--------------------	---------

Issue Links:
[Manage Links](#)

Story/Task Relationship

This issue contains:

- [PR-2](#) S01-T01 Extract hard coded configurat...
- [PR-40](#) S01-T02 Konfiguration zentralisieren
- [PR-42](#) S01-T05 Dokumentation
- [PR-45](#) S01-T03 provide SWT-Bundles
- [PR-46](#) S01-T04 OSGi Knopf. Run Configuration
- [PR-24](#) S01-T06 Rename BundleSWT to BundleSWT...
- [PR-26](#) S01-T07 Move native-code libraries fr...
- [PR-34](#) S01-T08 SPIKE: automatic MAC address ...
- [PR-36](#) S01-T09 Create empty files if missing
- [PR-38](#) S01-T10 Discuss name and responsibili...
- [PR-38](#) S01-T11 Write a short tutorial how to...
- [PR-30](#) S01-T12 Check the documentation for u...

Description

It shall be as simple as possible to checkout the sources of the project and run the bundles. Everything a developer needs to know for configuration is clearly documented in our wiki (<http://roots.iai.uni-bonn.de/prompter/>). Dependencies on the J9 and eSWT shall be elimin... unless they need a global restructuring. All configuration informations that are still hard coded at the moment shall be extracted to a configuration file.

Issue Tracking System „Jira“: Tasks

http://roots.iai.uni-bonn.de/jira32/browse/PR-2

ROOTS
Research on Object-Oriented Technologies and Systems

User: Daniel Speicher [History](#) | [Filters](#) | [Profile](#) | [Log Out](#)

[HOME](#) [BROWSE PROJECT](#) [FIND ISSUES](#) [CREATE NEW ISSUE](#) [ADMINISTRATION](#) [QUICK SEARCH:](#)

Issue Details [\[XML\]](#)

Key: [PR-2](#)

Type: Task

Status: Closed

Resolution: Fixed

Priority: Major

Assignee: [Kai-Lin Pang](#)

Reporter: [Daniel Speicher](#)

Votes: 0 [\(New\)](#)

Watchers: 0 [\(New\)](#)

Available Workflow Actions

☐ [Reopen Issue](#)

Operations

☐ [Assign](#) this issue [\(to me\)](#)

☐ [Clone](#) this issue

☐ [Comment](#) on this issue

☐ [Delete](#) this issue

☐ [Link](#) this issue to another issue

☐ [Move](#) this issue

☐ [Watch](#) this issue

Prompter

S01-T01 Extract hard coded configuration information

Created: 02/Nov/05 02:31 PM Updated: 05/Nov/05 05:03 PM

Component/s: None

Affects Version/s: None

Fix Version/s: [0.2M1](#)

[Return to search](#)
"Prompter stories"

Original Estimate:	Unknown	Remaining Estimate:	Unknown	Time Spent:	Unknown
---------------------------	---------	----------------------------	---------	--------------------	---------

Issue Links: [Manage Links](#)

Story/Task Relationship

This issue is part of:

[PR-4](#) S01 Easy Checkout

Description

Create a class in the BundleCSCommon to access a properties file (or xml configuration file). Replace all references to hard coded configuration information to an access to this class.

[All](#) [Comments](#) [Work Log](#) [Change History](#) [Sort](#)

Comment by [Kai-Lin Pang](#) [05/Nov/05 05:03 PM] [Delete](#)

All important hard coded references (ONLY Bundles) are replaced by a properties file called "config.ini" in BundleCSCommon.

Werkzeuge: Story Tracking als Beispiel

- Papier
 - ◆ (+) Besserer Überblick über den Gesamtzustand
 - ◆ (+) Leichter zu modifizieren.
 - ◆ (+) Ein Bild sagt mehr als tausend Worte.
 - ◆ (-) Verschwinden nach dem Kurs.
 - ◆ (-) Keine Anfragen möglich.

- Jira (Web-basiertes Issue Tracking)
 - ◆ (+) Persistente Repräsentation der Entwicklungsgeschichte
 - ◆ (+) Lange Zeit sichtbar
 - ◆ (-) Überblick nicht so klar
 - ◆ (-) Eintragen der Tasks / Stories aufwändiger, keine Zeichnungen

Eine genauere Betrachtung

Einflussfaktoren auf den SW-Entwicklungsprozess

- Kosten
- Qualität
- Zeit
- Funktions-Umfang

Grund-Zusammenhang

Durch Festlegung von drei beliebigen Faktoren ist der Vierte mit festgelegt !

Konsequenz

Der Kunde kann höchstens drei Faktoren nach seinem Wunsch bestimmen.
Die Entwickler müssen ihm den Einfluss auf den vierten Faktor erläutern!

Einfluss des Kostenrahmens

- zu wenig Geld
 - ◆ keine effektive Entwicklung
- mehr Geld
 - ◆ bessere Ausstattung, Umgebung, Ausbildung, ...
- aber Geld allein macht kein erfolgreiches Projekt
 - ◆ "40 Programmierer"-Anekdote
- besser
 - ◆ Projektgröße schrittweise anpassen
- Problem
 - ◆ Status- / Prestige-Denken von Projektleitern
 - ⇒ "Ich hab ein 150-Personen-Projekt..."

Einfluss des Zeitrahmens

- zu wenig Zeit
 - ◆ schlechte Qualität
 - ◆ geringer Umfang
 - ◆ hohe Kosten
- mehr Zeit
 - ◆ bessere Qualität
 - ◆ mehr Funktionalität
- aber zu viel Zeit bis zur Kundenpräsentation / Inbetriebnahme schadet
 - ◆ kein Feedback aus laufendem Betrieb
 - ◆ ... das ist aber das wertvollste Feedback überhaupt
- wenig Einflussmöglichkeiten durch Programmier-Team
 - ◆ Zeitrahmen ist meist vom Kunden bestimmt
 - ⇒ Jahr 2000-Problem, Euro-Umstellung, nächste Messe, ...

Einfluss der Qualität

- externe Qualität
 - ◆ was der Kunde sieht: Funktionalität, Geschwindigkeit, Zuverlässigkeit, ...
 - interne Qualität
 - ◆ was der Entwickler sieht: Code-Struktur, Wartbarkeit, Verständlichkeit, ...
 - geringere interne Qualität
 - ◆ kurzfristige Zeitersparnis
 - ◆ langfristige Wartbarkeitskatastrophe
 - ◆ langfristig schlechte externe Qualität
 - ◆ demoralisierender Effekt im Team
 - hohe interne Qualität
 - ◆ langfristig schnellere Entwicklung
 - ◆ dauerhaft gute externe Qualität
 - ◆ bessere Motivation / Zufriedenheit des Teams
 - ◆ Kundenzufriedenheit
- ➔ wenig Spielraum!

Einfluss des Funktions-Umfangs

- Wichtigste Einflussmöglichkeit (laut Kent Beck)
- Idee
 - ◆ Kosten, Zeit und Qualität festlegen
 - ◆ realisierbaren Funktionsumfang ermitteln
- Vorteile
 - ◆ leichte Anpassbarkeit an Änderungsanforderungen
- Risiko
 - ◆ zu viele / zu wichtige Funktionen werden gestrichen
- XP-Ansatz
 - ◆ Wichtigste Kundenanforderungen zuerst realisieren
 - ⇒ nur Funktionalität mit geringer Priorität wird evtl. nicht realisiert
 - ◆ Aufwandsschätzungen mit Feedback
 - ⇒ bessere Schätzungen bedeuten weniger unrealisierbare Dinge

Einflussfaktoren: Fazit / Empfehlungen

- Kosten → Ressourcen bedarfsgerecht einsetzen
- Zeit → keine Einflussmöglichkeit, da extern bestimmt!
- Qualität → hohe interne Qualität ansreben!

bestimmen

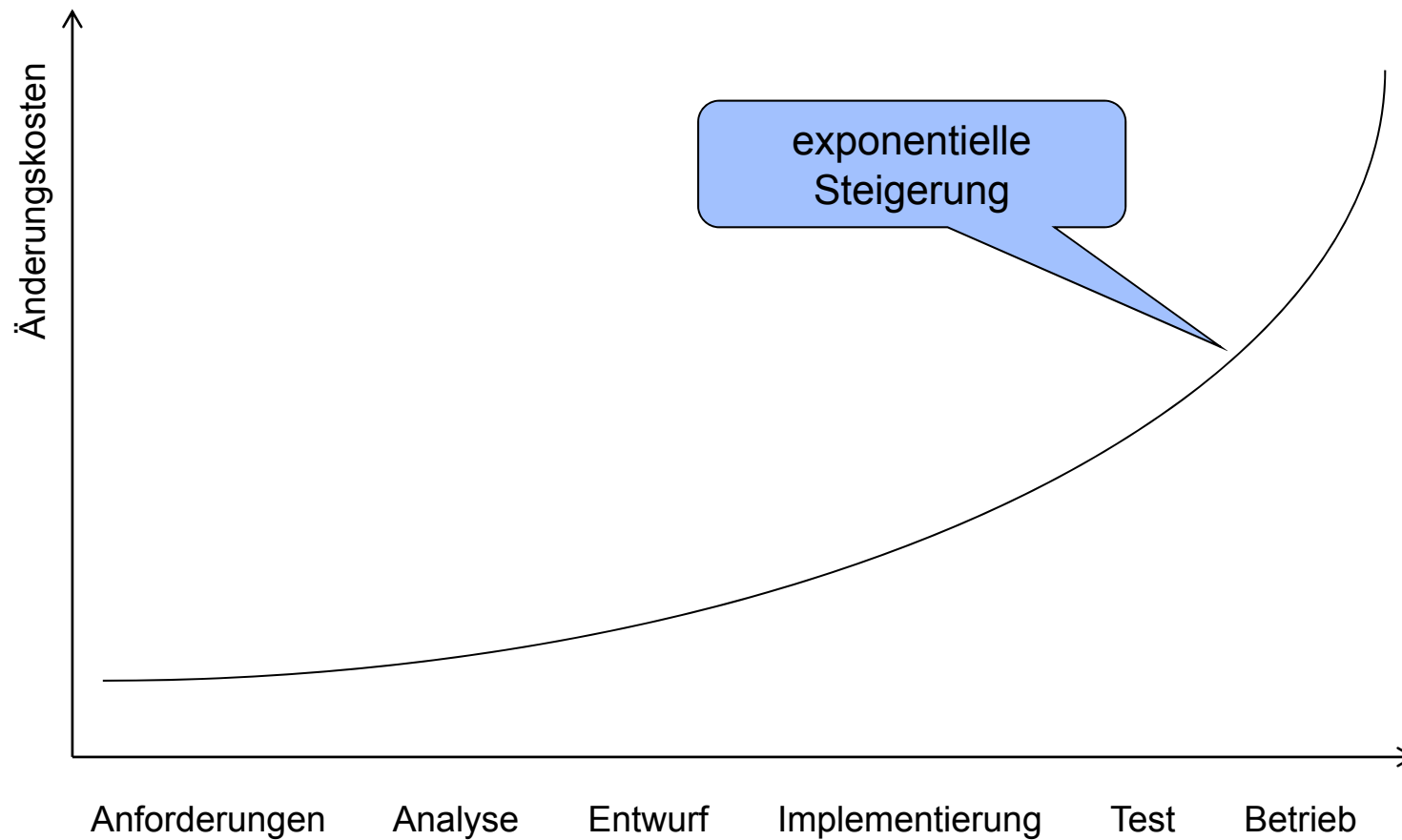


- Umfang → minimalen Funktions-Umfang anstreben!

**"So einfach wie möglich" klingt gut –
Aber: Was ist mit nachträglichen
Änderungen?**

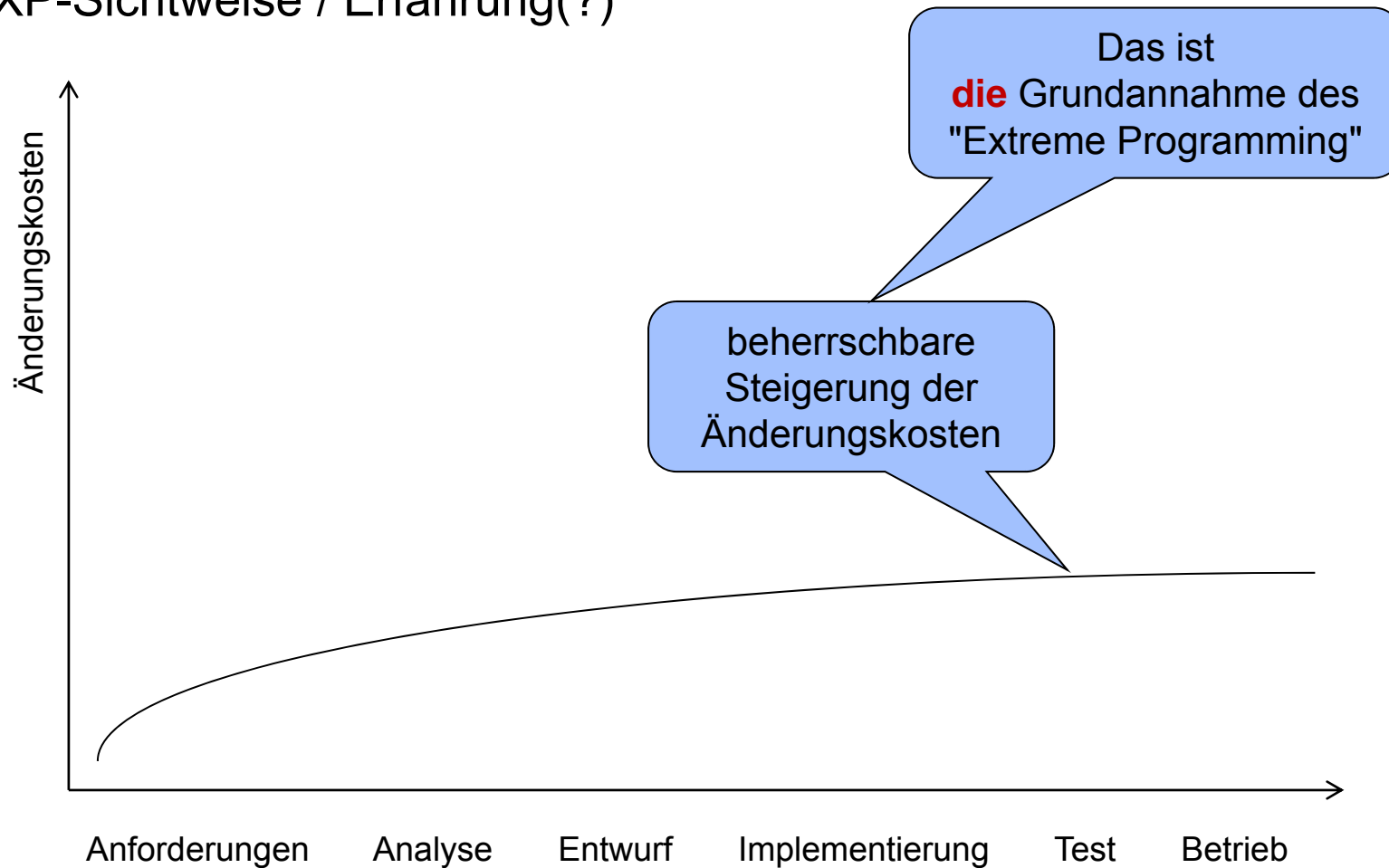
Wie teuer sind Änderungen?

- Traditionelle Sichtweise / Erfahrung



Wie teuer sind Änderungen?

- XP-Sichtweise / Erfahrung(?)



Gegenüberstellung

Traditionelle Methodiken

Extreme Programming

- Annahme
 - ◆ exponentielle Kostenexplosion
 - ◆ asymptotische Kostensteigerung
- Konsequenzen
 - ◆ mögliche Änderungen antizipieren
gefordert
 - ◆ Änderungen erst bedenken wenn
 - ◆ entsprechende Möglichkeiten
einbauen
 - ◆ nur das notwendigste
Implementieren
 - ◆ komplexere Software
 - ◆ einfachere Software
 - ◆ höhere Anfangskosten
 - ◆ geringere Anfangskosten
 - ◆ langsamerer Anfangsfortschritt
 - ◆ schnellerer Projektfortschritt
 - ◆ geringere Gesamtkosten!!!
 - ◆ geringere Gesamtkosten!!!
 - ◆ geringere Gesamtzeit!!!
 - ◆ geringere Gesamtzeit!!!
- Fragen
 - ◆ welche Annahme stimmt?
 - ◆ was würde eine asymptotische Steigerung plausibel machen?

Wege zur Reduktion der Änderungskosten

- Einsatz objekt-orientierter Systeme
- Einfaches Design
- Automatische Tests
- Erfahrung im Verändern

Das macht asymptotische Änderungskosten plausibel!

Macht das asymptotische Änderungskosten plausibel?

Die Lösungs-Idee: "XP ist wie Autofahren"

"XP ist wie Autofahren"

- So einfach?
- So unverzichtbar?
- Nein, sondern
 - ◆ es erfordert dauernde Aufmerksamkeit
 - ◆ es erfordert ständige kleine Richtungskorrekturen

"Driving is not about getting the car going in the right direction.
Driving is about **constantly paying attention**,
making a **little correction** this way, a **little correction** that way."

"XP erfordert nicht weniger Disziplin als andere Methodiken.
Blos die Dinge die als wichtig erachtet werden und Disziplin erfordern sind andere."

Planspiel

- SW-Entwicklung als Dialog zwischen
 - ◆ Wünschenswertem (Kunden-Sicht)
 - ◆ Machbarem (Programmierer-Sicht)
- Kunden entscheiden über
 - ◆ Funktionsumfang
 - ◆ Prioritäten
 - ◆ Zusammensetzung von Releases die einen echten Mehrwert bietet
 - ◆ Zeitpunkt von Releases
- Programmierer entscheiden über
 - ◆ Aufwandsschätzungen
 - ◆ Technische Folgen eines Kundenwunsches (z.B. DB-Auswahl)
 - ◆ Prozess (interne und externe Team-Zusammenarbeit)
 - ◆ Interne Zeitpläne
 - ⇒ was passiert innerhalb eines Release zuerst?
 - ⇒ risiko-behaftete Aspekte vorziehen

Pair Programming

- Szenario
 - ◆ ein Rechner, eine Tastatur, eine Maus
 - ◆ zwei Programmierer
 - ◆ Rollen-Teilung
- Implementierer-Rolle
 - ◆ implementiert
- Code-Reviewer-Rolle
 - ◆ Kann das so funktionieren?
 - ◆ Gibt es Testfälle, die wir noch nicht bedacht haben?
 - ◆ Kann man das Problem durch eine Vereinfachung des Designs lösen?
 - ◆ ...
- Rollen jederzeit änderbar
- Paare jederzeit änderbar (tyischerweise nach Erledigung einer Task)

XP-Management (1)

- Coach (Betreuer)
 - ◆ "Ein guter Lehrer macht sich selbst langfristig überflüssig".
 - ◆ entscheidet nicht, sondern
 - ◆ ... hilft anderen gute Entscheidungen zu treffen
 - ◆ implementiert selbst wenig sondern
 - ◆ ... ist als "pair programmer" für andere (Neulinge) verfügbar
 - ◆ ... sieht langfristige Refactorings voraus und ermutigt dahingehend
 - ◆ ... hilft in speziellen technischen Bereichen
 - ◆ erklärt den Prozess dem Management

- Metriken
 - ◆ direktes Feedback über Projektzustand
 - ◆ Bsp: "Project Velocity" (Verhältnis zwischen Schätzung und Realität)
 - ◆ "Big Visible Chart"
 - ◆ nicht mehr als drei Metriken

XP-Management (2)

- Tracking
 - ◆ Planspiel
 - ◆ Metriken zur Verifikation (zwei Messungen pro Woche reichen)
 - ◆ Anpassungen des Plans

- Eingriff
 - ◆ Wenn's sein muss auch unpopuläre Korrekturen entscheiden hinsichtlich
 - ⇒ Funktionsumfang
 - ⇒ Architektur
 - ⇒ Team-Zusammensetzung
 - ◆ "Humility is the rule of the day for an intervention."

XP Rollen: Kunde

- Spezifiziert:
 - ◆ Funktionale und nicht funktional Anforderungen → „Stories“
 - ◆ Priorisiert die Stories
- Ist verfügbar für klärende Fragen des Teams bezüglich:
 - ◆ Unvollständiger, unklarer oder inkonsistenter Anforderungen
 - ◆ Des relevanten Geschäftsprozesses
 - ◆ Des Anwendungsgebietes

XP Rollen: Kunde (2)

- Entscheidet über:
 - ◆ Die Menge an Funktionalität (Stories) die in der nächsten Iteration / Release zu implementieren sind, um den Anwendern einen wirklichen Nutzen zu bringen
 - ◆ Die Entscheidung basiert auf der vorangegangenen Diskussion mit dem Team („Planning Game“) um ihre Realisierbarkeit und Auswirkungen abschätzen zu können
- Führt an den vorhandenen Releases Systemtests durch
 - ◆ Funktionale Tests („Tut es das was es soll?“)
 - ◆ Acceptance Tests („Tut es dies auf eine Art und Weise, die von den Anwendern einfach und intuitiv benutzt werden kann?“)
- Gibt dem Team Feedback über die getesteten Releases

XP Rollen: Programmierer (inkl. Teamleiter, ...)

- Führen durch
 - ◆ Schätzungen der Zeit / Schwierigkeit und des Risikos der Stories
 - ◆ Unterteilung der Stories in Tasks
 - ◆ Schätzung der Task - Implementationsdauer
 - ◆ Priorisieren die Tasks
- Verpflichten sich
 - ◆ eine Task auszuführen
 - ◆ Kollegen zu helfen wenn nötig
- Entscheiden über
 - ◆ Technische Folgen der Kundenanforderungen (z.B. Wahl eines DBMS)
 - ◆ Den Prozess (wie das Team arbeitet und sich selbst organisiert)
 - ◆ Internes Zeitmanagement
 - ⇒ Was passiert während einer Iteration zuerst?
 - ⇒ Erledige die Tasks mit dem höchsten Risiko zuerst!

XP Rollen: Teamleiter

- Kontrolliert den Prozess
 - ◆ Leitet das Team durch den täglichen Ablauf
 - ◆ Behält den Fortschritt des Teams im Auge
 - ◆ Vergleicht ihm mit den Schätzungen (z.B. mit „Burn-Down Charts“)
 - ◆ Gibt dem Team Feedback über das Verhältnis von Schätzung zu Realität
 - ◆ Identifiziert Probleme oder „Flaschenhälse“ im Voraus
 - ◆ Denkt über benötigte Änderungen nach (Prozess, Ablauf, Teams, ...)
- Vermittelt die Kommunikation mit dem Kunden
- Erklärt dem Kunden die Folgen seiner Anforderungen
 - ◆ Erklärt dem Kunden auftretende Probleme und regt Diskussionen über notwendige Zurückstellung von Stories in die nächste Iteration an.

XP Rollen: Teamleiter (2)

- Leitet die Diskussionen des Teams und erklärt Diskussionstechniken (wenn nötig)
- Zielgerichtete Diskussion
 - ◆ Konstruktive Vorschläge
 - ◆ Den Kollegen zuhören
 - ◆ Klare Ergebnisse
 - ◆ Verpflichtungen: Was muss wann von wem getan werden?

XP Rollen: Technischer Experte („Consultant“)

- Ist Experte auf einem für das Projekt wichtigen Feld
- Ist in der Lage seine Expertise dem Team zu vermitteln
 - ◆ Präsentationen
 - ◆ Tutorials
 - ◆ Pair Programming mit Teammitgliedern
 - ◆ Beobachten und beraten von eigenständig arbeitenden Teammitgliedern
 - ◆ Macht Code Reviews
- Muss verfügbar sein wenn er benötigt wird
 - ◆ Gut wenn er dauerhaft vor Ort ist (aber nicht nötig)
 - ◆ Ausreichend wenn er sich in der Nähe aufhält und für das Team verfügbar ist wenn er gebraucht wird

XP Rollen: XP Mentor

- Hat Erfahrung in XP Techniken und Praktiken
- Kann das Team durch den XP Prozess führen
 - ◆ Erklärt XP Techniken und Praktiken
 - ◆ Erklärt ihren Wert und die Auswirkungen wenn man ihnen nicht folgt
 - ◆ Überwacht den Prozess und zeigt Techniken auf die
 - ⇒ nicht angewendet werden
 - ⇒ nicht effektiv angewendet werden
 - ⇒ falsch angewendet werden
 - ◆ Überwacht den Prozess und macht Aufmerksam auf
 - ⇒ nicht-XP Elemente
 - ⇒ anti-XP Elementeund erklärt ihre Gefahren sowie XP-Style Alternativen

Agile Methodologien - Lehre

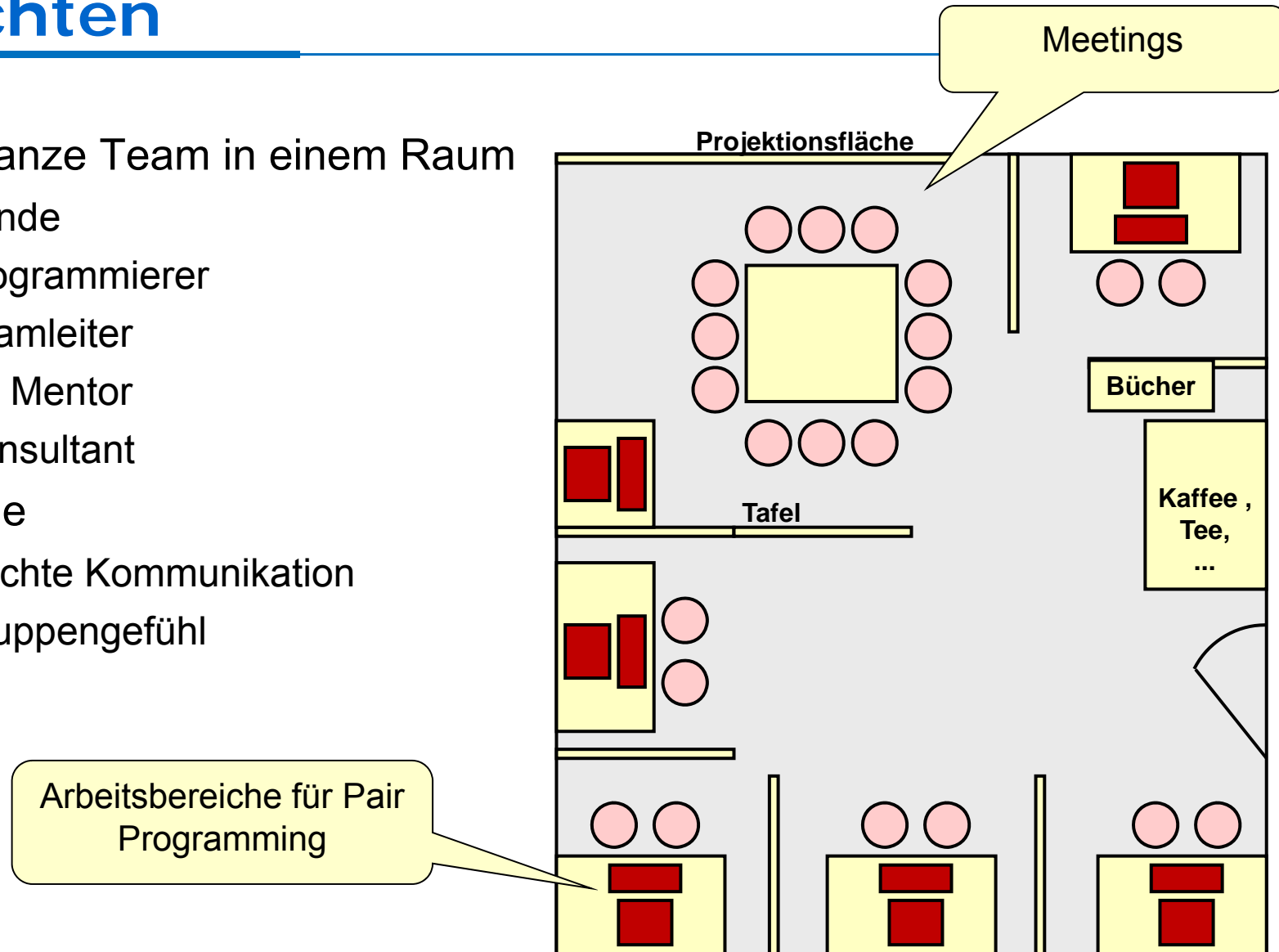
- Praktische XP Kurse am B-IT -

XP Lehre: Rollen

<ul style="list-style-type: none">● Programmierer<ul style="list-style-type: none">◆ schätzen, planen, designen, implementieren, testen	Studenten
<ul style="list-style-type: none">● Teamleiter<ul style="list-style-type: none">◆ leitet die Programmieren● Consultant<ul style="list-style-type: none">◆ technischer Experte auf einem bestimmten Gebiet● XP Mentor<ul style="list-style-type: none">◆ XP Experte der das gesamte Team berät	Dozenten
<ul style="list-style-type: none">● Kunde<ul style="list-style-type: none">◆ entscheidet über Funktionalität	Wirklicher Kunde oder Dozent

XP Lehre: Einen geeigneten Raum einrichten

- Das ganze Team in einem Raum
 - ◆ Kunde
 - ◆ Programmierer
 - ◆ Teamleiter
 - ◆ XP Mentor
 - ◆ Consultant
- Vorteile
 - ◆ Leichte Kommunikation
 - ◆ Gruppengefühl



XP Lehre: Die Praktika

- Intensivkurse
 - ◆ 4 bis 6 Wochen
 - ◆ 8 Stunden pro Tag!
- Gute Betreuung
 - ◆ 2-4 Dozenten für 10-16 Studenten
- Professionelle Arbeitsumgebung
 - ◆ Eigenes Büro
 - ◆ Moderne Ausrüstung (Computer, Beamer)
 - ◆ Aktuelle Tools (Eclipse, SVN, Jira, Greenhopper, “Touchscreen” auf Wand, Integrationsserver, Commit-Ampel, Wikis, ...)
- Interessante und realistische Projekte
 - ◆ Aufgaben sind Teil von Forschungsprojekten