Warum XP?
Methoden
Beispiele

eXtreme Programming

Konzepte, Ziele, Methoden

Praktische Erfahrungen aus XP-Projekten

Ein Vortrag im Rahmen des Seminars
„Refactoring in eXtreme Programming“
von Björn Zeiger

Universität Paderborn, AG Kastens
Januar 2003

Warum XP?
Methoden
Beispiele

Ablauf:

- Warum XP?
- Konzepte und Methoden in XP
- Beispiele aus der Praxis

In diesem Abschnitt:

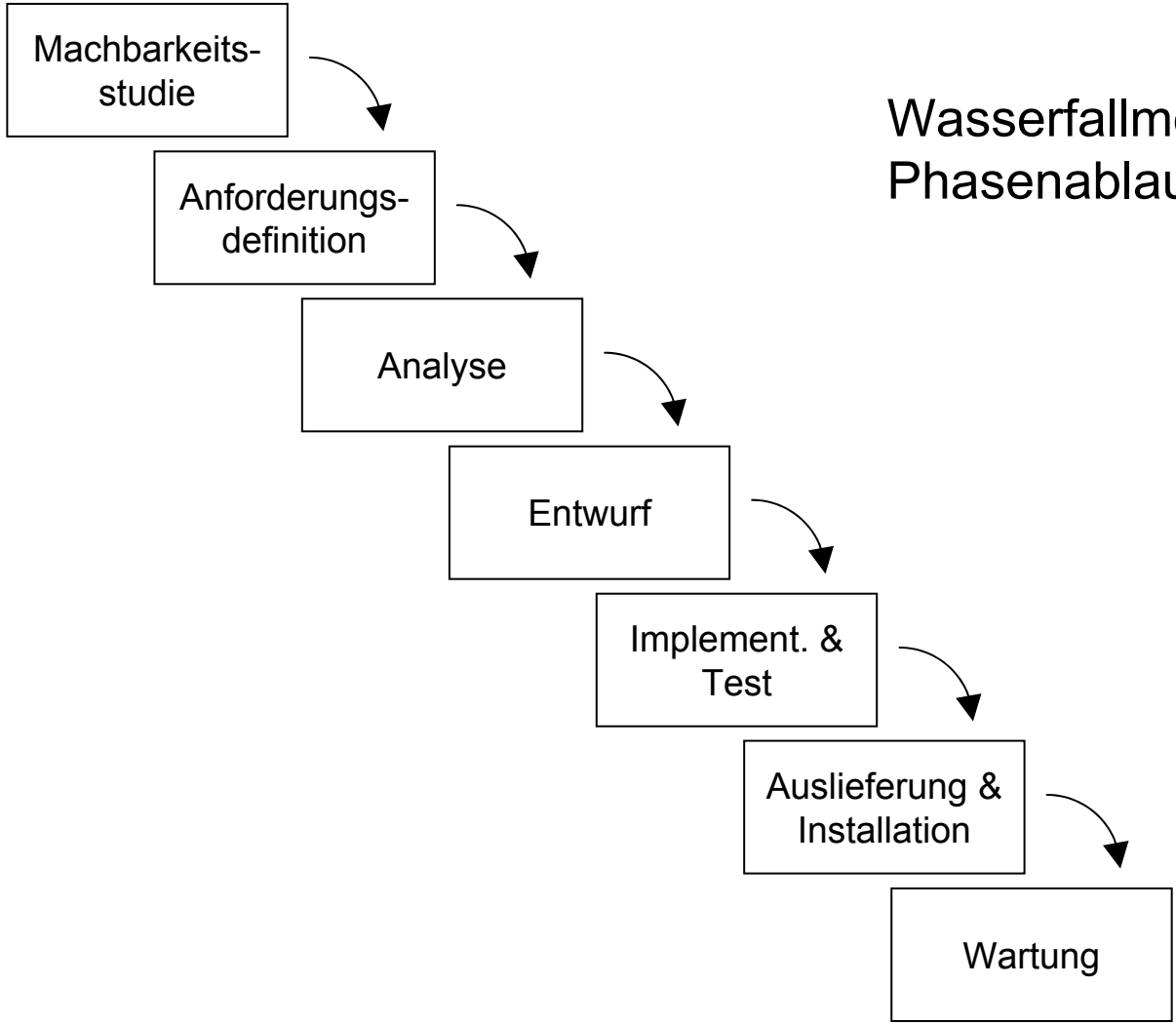
- Eigenschaften von XP
- Unterschiede zur klassischen Softwareentwicklung
- Wann soll/kann XP eingesetzt werden?

- Leitbild: Kommunikationsprozess
- Anforderungen in Interaktion mit der Software
- Anforderungsanalyse am "lebenden Programm"
- frühe Produktivschaltung, kurze Zyklen
- Vertrauensbildung durch schnelle Umsetzung von Kundenwünschen am laufenden Programm
- Entwicklungs-Risiko wird als unvermeidlich akzeptiert

Unterschiede zur „klassischen“ SE

- Leitbild: Ingenieur-Wissenschaft
- Festlegung der Anforderungen am Anfang
- Anforderungsanalyse in einem formal aufgebauten Pflichtenheft
- späte Produktivschaltung, lange Zyklen
- Vertrauensbildung durch Zertifizierungen und formale Qualitätssicherungsprozesse
- Versuch, Entwicklungs-Risiko auszuschließen

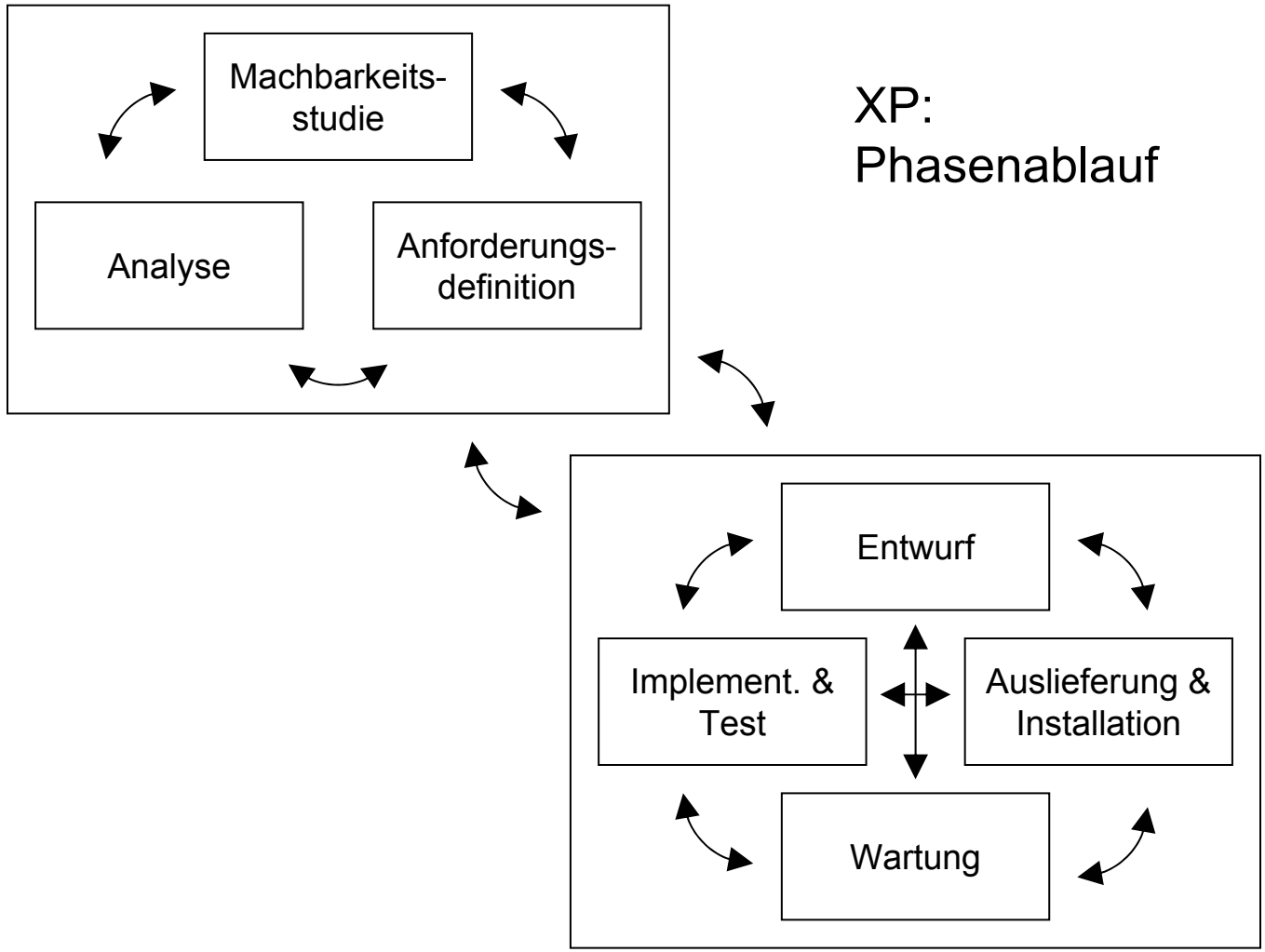
Unterschiede zur „klassischen“ SE



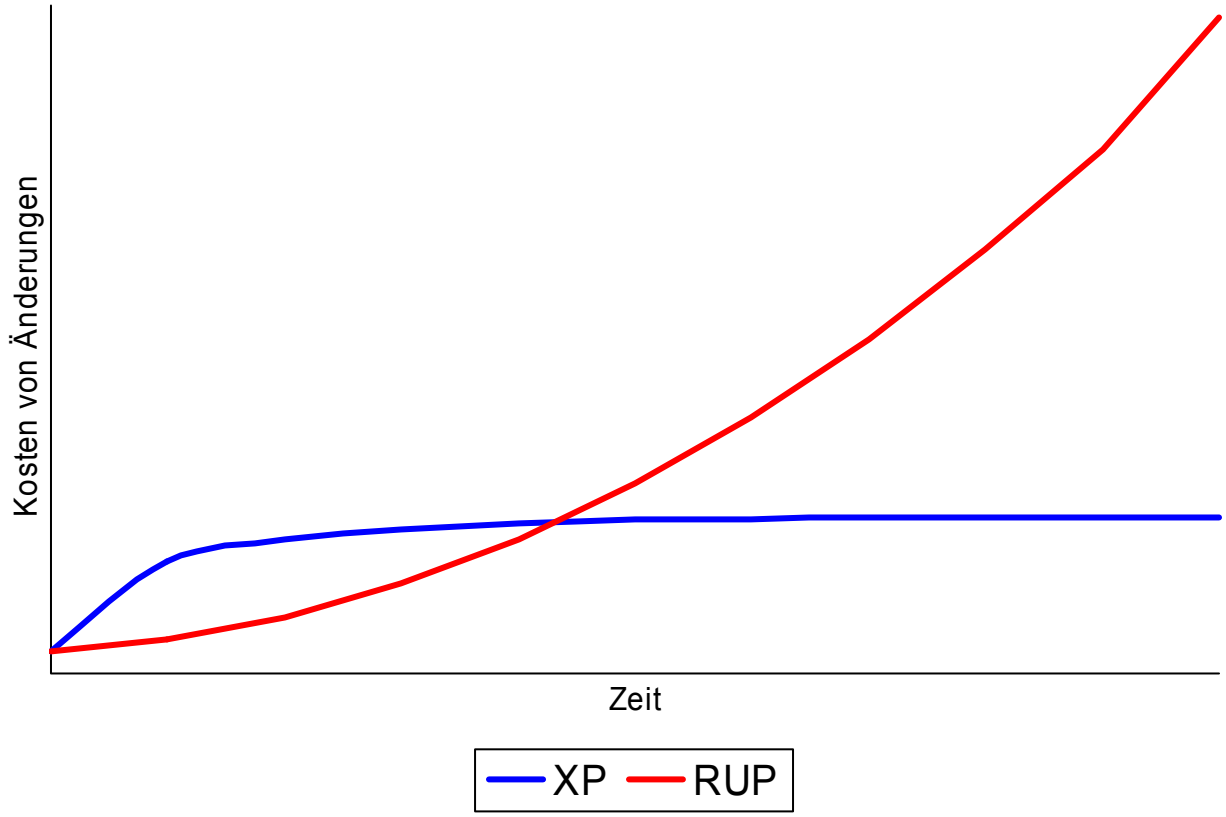
Wasserfallmodell:
Phasenablauf

Unterschiede zur „klassischen“ SE

XP:
Phasenablauf



Kosten von Änderungen



Wann sollte XP eingesetzt werden?

„In many software environments dynamically changing requirements is the only constant.“
(Don Well auf www.extremeprogramming.org)

- Keine klaren Vorstellungen über Funktionalität
- Sich ständig ändernde Anforderungen
- Hohes Projektrisiko (Erfahrung/Fristen)
- Direkte Einbeziehung des Kunden von Anfang bis Ende
- Kleines Entwicklerteam (K.B.: 2-10 Personen)

Warum XP?
Methoden
Beispiele

In diesem Abschnitt:

- Übersicht über Konzepte und Methoden
- Vorstellung im einzelnen
- Abhängigkeiten

Konzepte & Methoden: Übersicht

Kunde vor Ort

Planungsspiel

40-Stunden-
Woche

Metapher

Refactoring

Einfaches Design

Kurze
Releasezyklen

Testen

Programmieren
in Paaren

Programmier-
Standards

Gemeinsame
Verantwortlichkeit

Fortlaufende
Integration

Konzepte & Methoden: Gliederung (1)

Warum XP?

Methoden

Beispiele

Kunde vor Ort

Planungsspiel

40-Stunden-Woche

Metapher

Refactoring

Einfaches Design

Kurze Releasezyklen

Testen

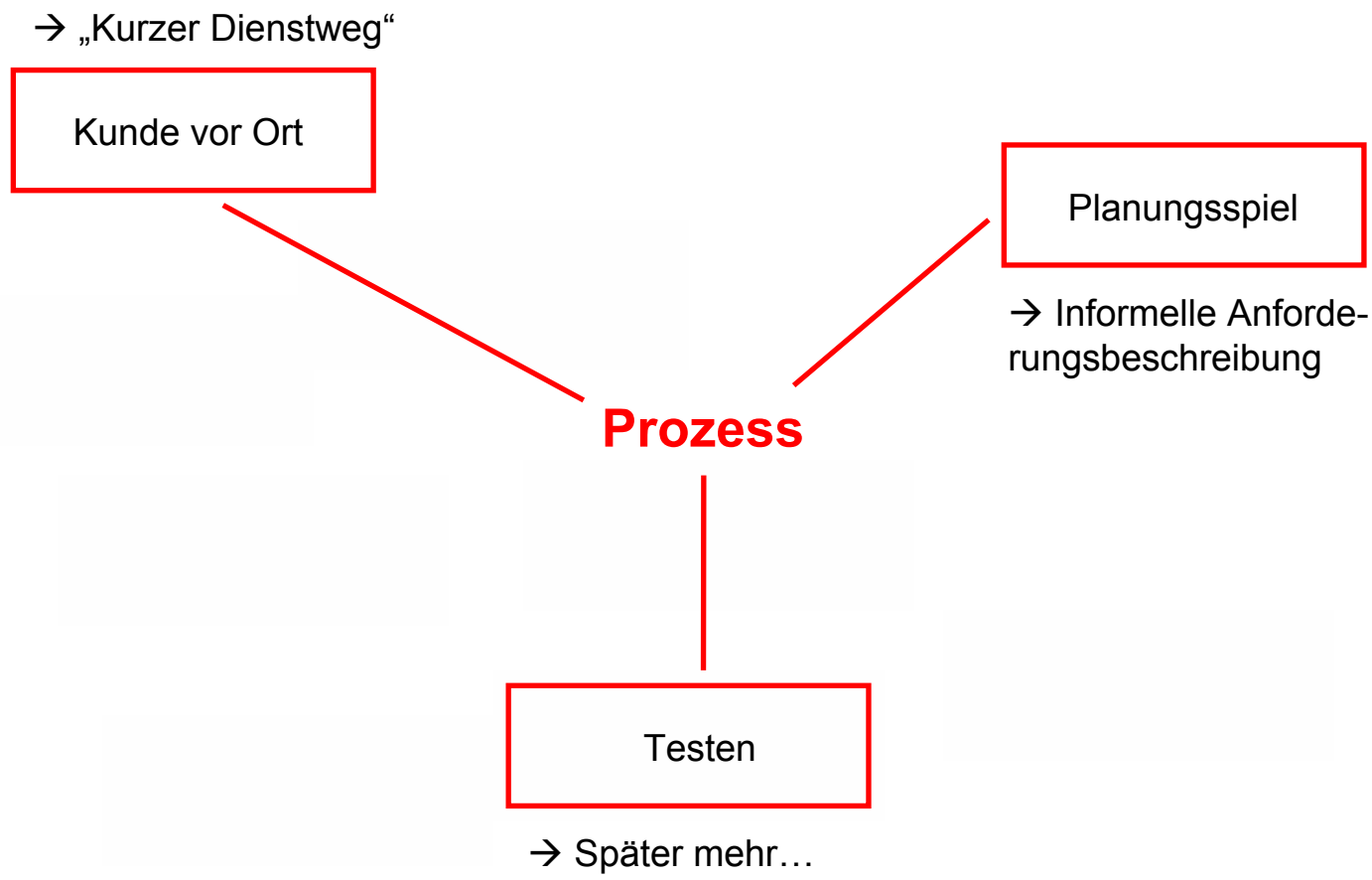
Programmieren in Paaren

Programmier-Standards

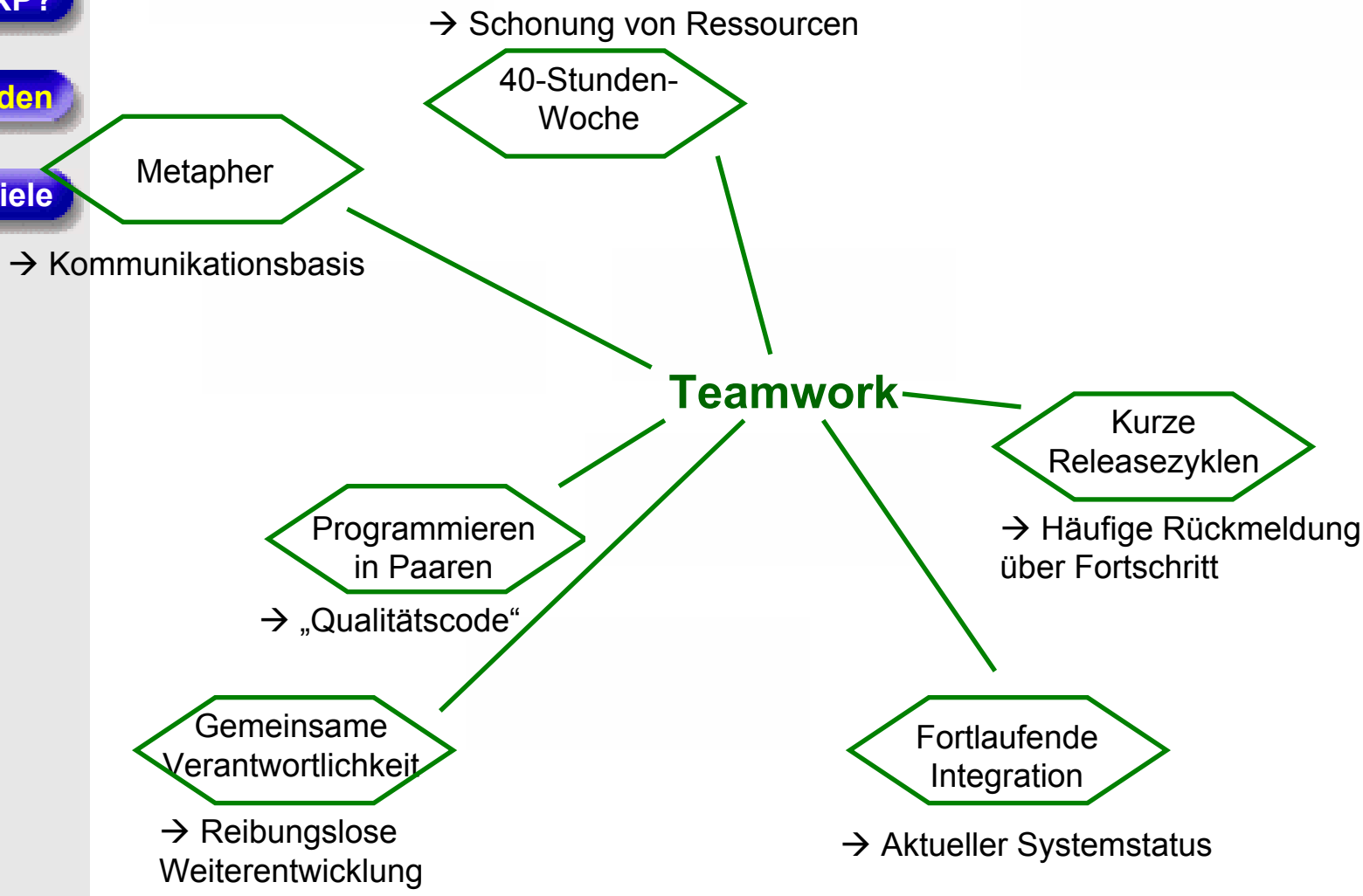
Gemeinsame Verantwortlichkeit

Fortlaufende Integration

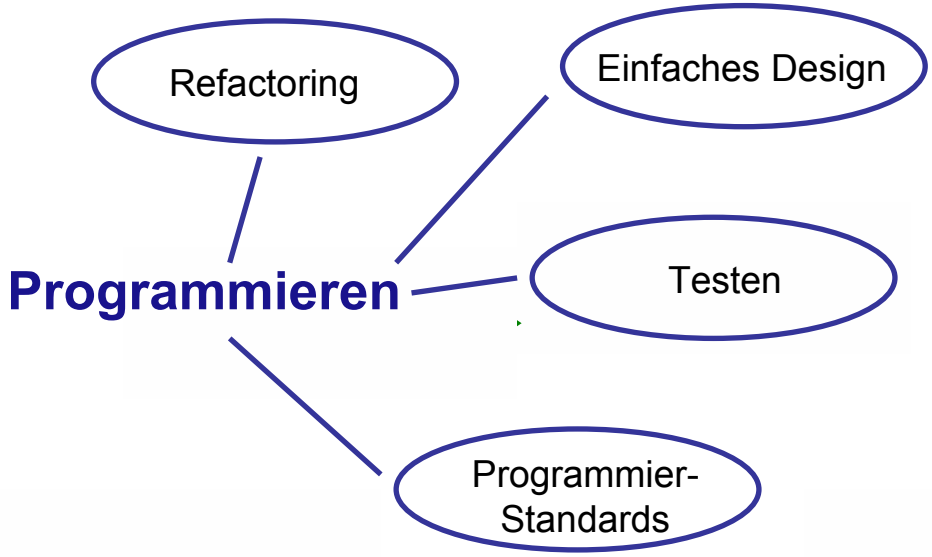
Konzepte & Methoden: Gliederung (2)



Konzepte & Methoden: Gliederung (3)



Konzepte & Methoden: Gliederung (4)



Ein motivierendes Beispiel…

- Zwei Programmierer
- Aufgabe: Währungsrechner Euro ⇔ DM

Programmierer A:

```
public class CurrencyCalculator {

    public static final  int    DM_TO_EURO        = 0;
    public static final  int    EURO_TO_DM        = 1;
    private static final double RATE_OF_EXCHANGE = 1.95583;

    private int mode;

    public CurrencyCalculator(int _mode) {
        mode = _mode;
    }

    public double caluclateValue(double value) {
        switch (mode) {
            case DM_TO_EURO:
                return value * RATE_OF_EXCHANGE;
            case EURO_TO_DM:
                return value / RATE_OF_EXCHANGE;
        }
        return -1d;
    }
}
```

Programmierer B:

```
public class Rechner {
public static final int de = 0;
public static final int ed = 1;
public double rechne(double v) {
switch (m)
{case de: return v*k;
case ed: return v/k;}
return -1d;}
private static final double k = 1.95583;
public Rechner(int _m)
{m = _m;}
private int m;
}
```

Standards für…

- Benennung von Klassen und Methoden
- Positionen von Variablen- und Methoden-Deklarationen
- Formatierung (Position von Klammern, Einrückungen, Leerzeichen)
- Sprache, Kommentare
- Aber auch: Entwicklungsumgebungen, Shortcuts, etc.

- *siehe auch: http://java.sun.com/docs/codeconv/*

Warum Testen?

- Stärkung des Vertrauens der Entwickler in ihre Arbeit
- Sicherheit für den Kunden

Testen in XP

- Allgemein: weitestgehend automatisierte Tests
- Ggf. Unterstützt durch Werkzeuge
- Unterscheidung: Komponententests und Akzeptanztests

Komponententests

- Werden vom Entwickler geschrieben
- Nach Änderung oder Neuerung werden alle Komponententests ausgeführt

Akzeptanztests

- Werden vom Kunden spezifiziert
- Zeigen, ob das System wie gefordert funktioniert
- z.T. manuelles Testen notwendig

Testen in 5 Schritten:

1. TestCase schreiben
2. Skelett implementieren
3. Test ausführen
4. Skelett vervollständigen
5. Test ausführen

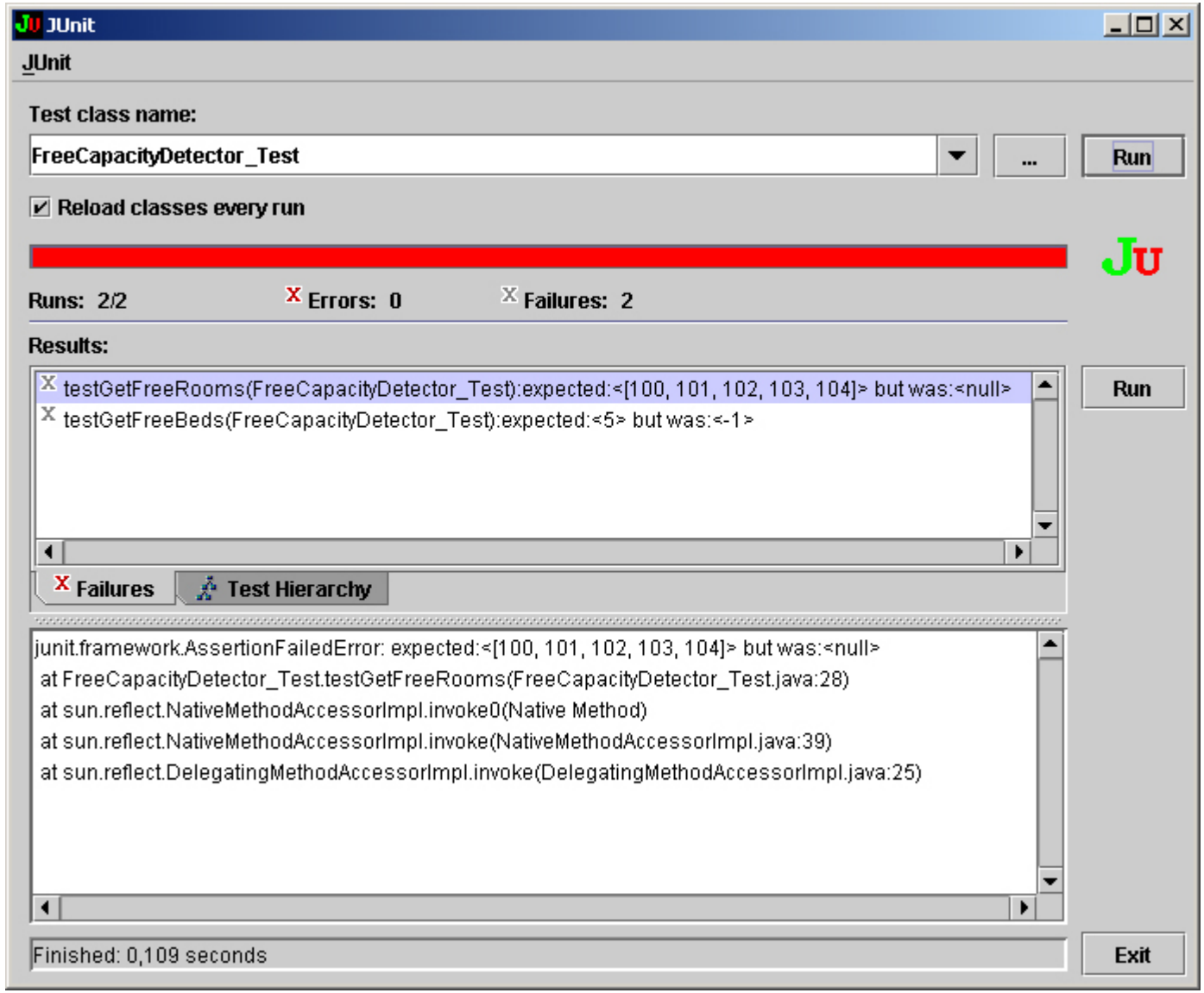
Freie Kapazitäten anzeigen (# 1202/7)
Nach Eingabe von zwei Daten prüft das System, welche Zimmer im gesamten Zeitraum frei sind und gibt eine Auflistung dieser aus.
Ebenfalls gezeigt werden soll die Anzahl der freien Betten im gewählten Zeitraum.

```
class FreeCapacityDetector_Test extends TestCase {

      private FreeCapacityDetector capDetector;
      private List freeRooms0 = …
      private List freeRooms1;
      private int freeBeds0 = …
      private int freeBeds1;

      public void testGetFreeCapacities() {
            freeRooms1 = capDetector.getFreeRooms();
            freeBeds1 = capDetector.getFreeBeds();
            assertEquals(freeRooms0, freeRooms1);
            assertEquals(freeBeds0, freeBeds1);
      }

      protected void setUp() {
            capDetector = new FreeCapacityDetector(
                              DateFormat.parse(„24.03.02“),
                              DateFormat.parse(„26.03.02“));
      }
}
```

Freie Kapazitäten anzeigen (# 1202/7)
Nach Eingabe von zwei Daten prüft das System, welche Zimmer im gesamten Zeitraum frei sind und gibt eine Auflistung dieser aus.
Ebenfalls gezeigt werden soll die Anzahl der freien Betten im gewählten Zeitraum.

```
class FreeCapacityDetector {

      private List freeRooms;
      private int freeBeds;

      public FreeCapacityDetector(Date dateFrom, Date dateTo) {}

      public List getFreeRooms() {
            return freeRooms;
      }

      public int getFreeBeds() {
            return freeBeds;
      }
}
```

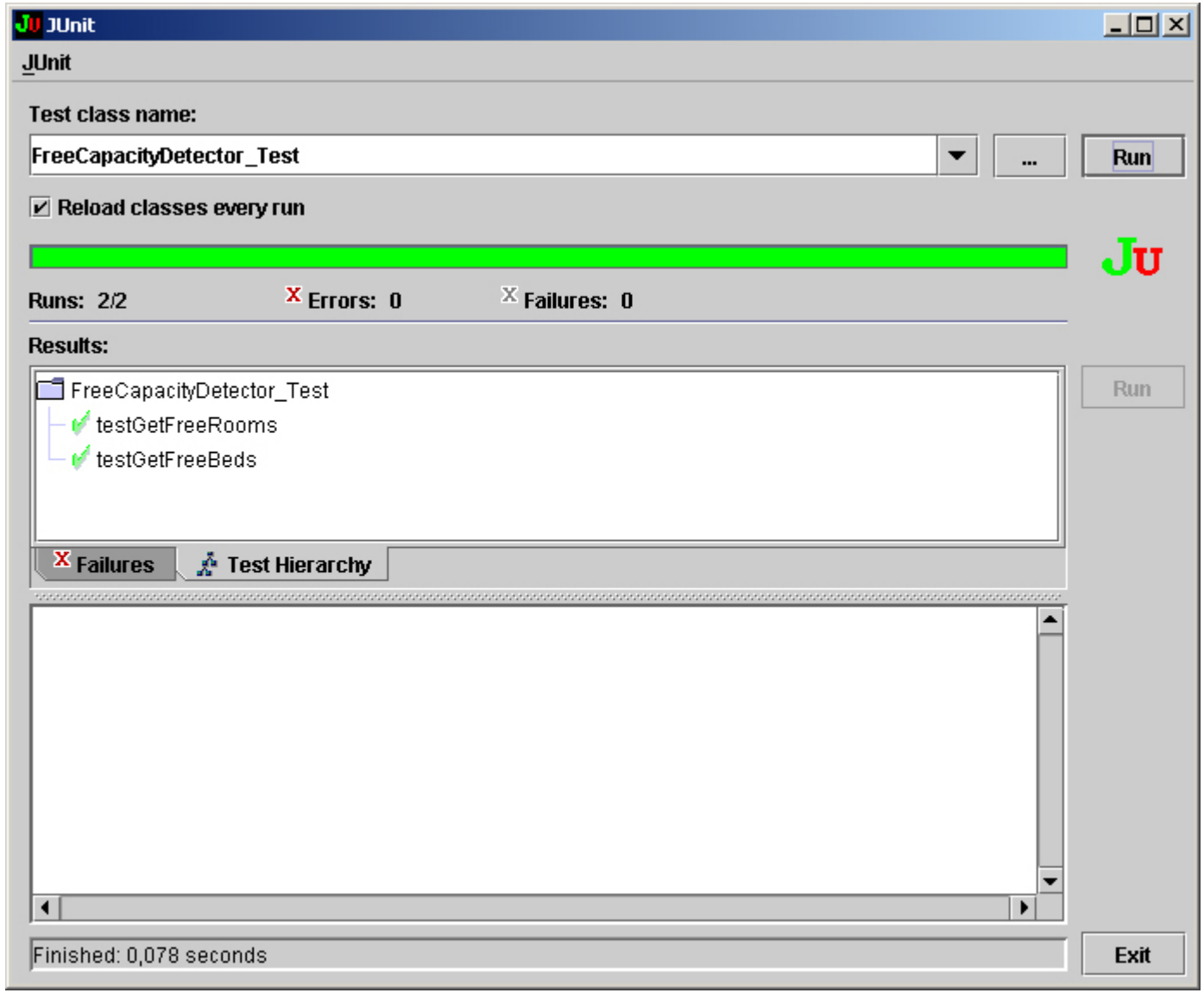
Testen/Schritt 3: Test ausführen



Freie Kapazitäten anzeigen (# 1202/7)
Nach Eingabe von zwei Daten prüft das System, welche Zimmer im gesamten Zeitraum frei sind und gibt eine Auflistung dieser aus.
Ebenfalls gezeigt werden soll die Anzahl der freien Betten im gewählten Zeitraum.

```
class FreeCapacityDetector {

      private List freeRooms;
      private int freeBeds;

      public FreeCapacityDetector(Date dateFrom, Date dateTo) {
            //do sth with these dates
      }

      public List getFreeRooms() {
            //do sth to fill freeRooms
            return freeRooms;
      }

      public int getFreeBeds() {
            //do sth to set freeBeds
            return freeBeds;
      }
}
```

Testen/Schritt 5: Test ausführen



Never believe in „Never change a running system“!
(aus: Software entwickeln mit XP)

- Änderungen im und am System
- Ohne Veränderung des beobachtbaren Verhaltens
- Zur Vereinfachung der Designs

- z.B. extrahieren von Methoden, Umbenennung von Variablen, Ändern von Parametern etc.
- Unterstützung durch zahlreiche Werkzeuge

Wann ist ein Design einfach?

1. Das System besteht alle Tests
2. Es enthält keine Redundanzen
3. Es spiegelt die Intention der Entwickler wieder
4. Es hat die geringste mögliche Anzahl von Klassen und Methoden

Konzepte & Methoden: Übersicht

Kunde vor Ort

Planungsspiel

40-Stunden-
Woche

Metapher

Refactoring

Einfaches Design

Kurze
Releasezyklen

Testen

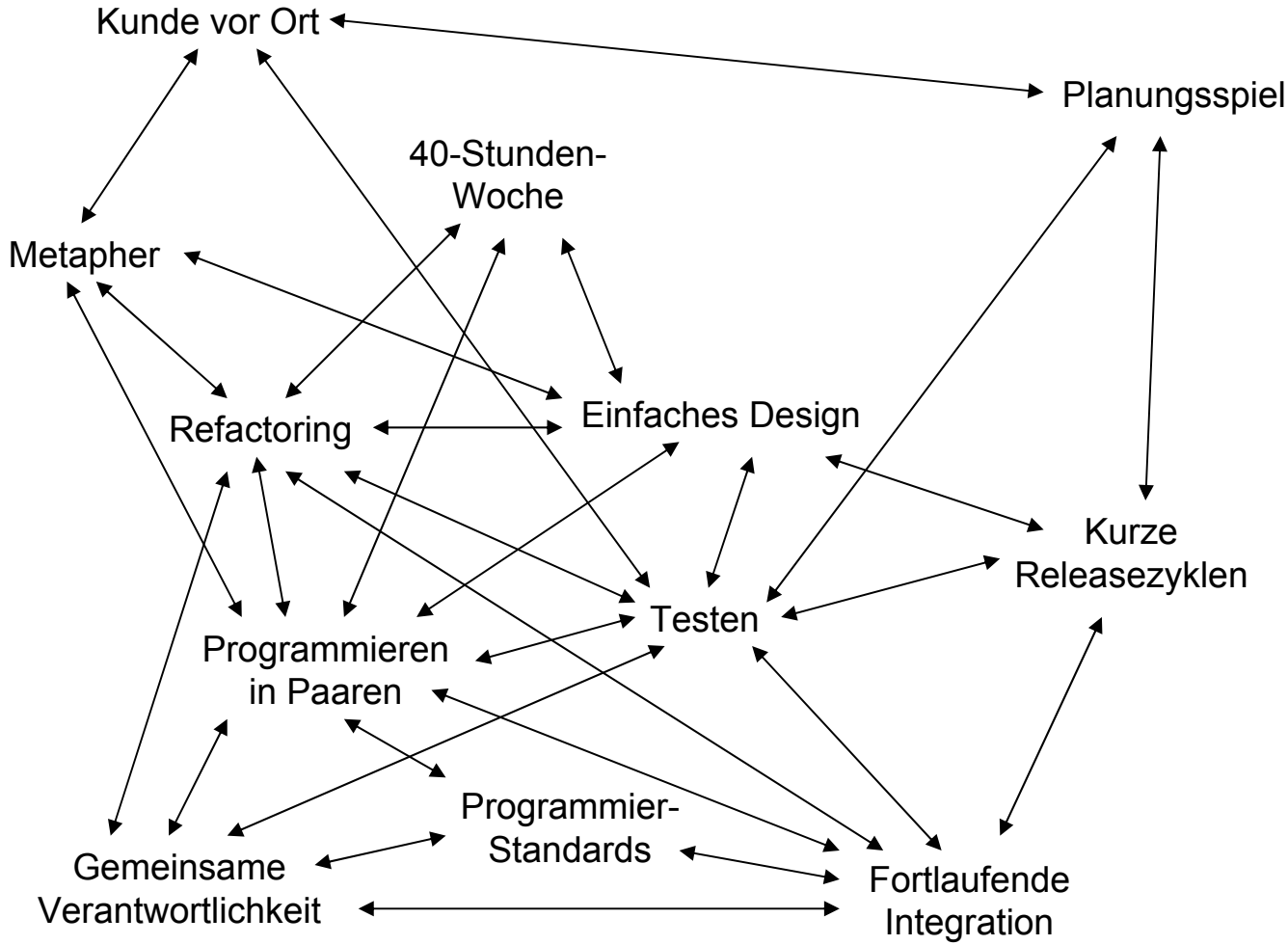
Programmieren
in Paaren

Programmier-
Standards

Gemeinsame
Verantwortlichkeit

Fortlaufende
Integration

Konzepte & Methoden: Abhängigkeiten



Warum XP?

Methoden

Beispiele

In diesem Abschnitt:

- Projektbericht „Kermit“
- Erfahrungen mit "Extreme Programming" im universitären Umfeld, Universität Karlsruhe

Warum XP?
Methoden
Beispiele

Kontext:

- Erweiterung des Großrechnersystems eines int. Dienstleistungsunternehmens
- Ziel: Abbildung länderspezifischer Vorgänge für KfZ-Rückkaufverträge mit komplexen Vertragsbedingungen

Ressourcen:

- Zwei Monate zwei Entwickler für zwei Tage/Woche zur Ermittlung der notwendigen Fachkompetenz
- Sechs Monate drei Entwickler für zwei bis drei Tage/Woche zur Entwicklung des Kernsystems

Warum XP?
Methoden
Beispiele

XP-Einsatz (1):

- Tests hilfreich zum „Ausprobieren“ von (Praxis-) Beispielen der komplexen Vertragsmodelle
- Programmieren in Paaren und gemeinsame Verantwortlichkeit kompensierten dreimonatigen Ausfall eines Entwicklers
- Schwierigkeit: „große“ Refactorings in genügend „kleine“ zerlegen
- Kein Planungsspiel aufgrund klarer Anforderungen

Warum XP?
Methoden
Beispiele

XP-Einsatz (2):

- Kunde vor Ort: ständige Verfügbarkeit per Telefon und Mail
- Einfache Kommunikation mit Kunden durch „passende“ Metapher: „durchscheinender Basisvertrag“
- Kurze Releasezyklen: erst Prototypen, um geminsame Vorstellungen über späteres System zu etablieren

Warum XP?
Methoden
Beispiele

Bewertung:

- Früher einsetzbare Versionen für schnellere Rückkopplung über Einsatz im Produktivbetrieb
- Insgesamt positive Einschätzung durch Kunden und Systemhaus
- Festpreis „harte Einschränkung“ für XP-Projekte

Erfahrungen mit "Extreme Programming"

Rahmenbedingungen:

- Teilnehmer: 12 (2 Gruppen á 6 Pers.)
- Qualifikation: Studenten im Hauptstudium
- Programmiersprache: Java
- Dauer: 12 Wochen
- Ablauf:
  Woche 1-3: „Fingerübungen“
  Woche 4-12: „Verkehrssimulation“

Erfahrungen mit "Extreme Programming"

Schwerpunkte:

- Programmieren in Paaren
- Iterationsplanung
- Testen
- Rafactoring
- Skalierbarkeit

Erfahrungen: Programmieren in Paaren

- Überwiegend als positiv bewertet
- Vorteil: vom Partner lernen (mit der Zeit abnehmend)
- Problem: Strukturierung der Arbeit
- Teilweise Benutzung von zwei PCs im Paar (Doku)
- Unklar, ob Vorteile wie höhere Qualität nicht durch weniger personalintensive Ansätze erreicht werden können (z.B. paarweise Code-Reviews)

Erfahrungen: Iterationsplanung

- Ansatz „highest priority first“ problematisch: Teilnehmer planen für Zukunft
- Minimaler Entwurf: Frage des Trainings oder schlechte Idee?
- „Entwurf mit Scheuklappen“

Erfahrungen: Testen

- Integration der Tests von Anfang an (relativ) problemlos
- Verzichten auf Test führte zu Problem: Team konnte nicht weiterarbeiten
- Verzicht auf automatische Tests der graphischen Darstellung (Zeitmangel)
- Als beste Praxis von XP bewertet
- Tests machen Teilnehmer sicherer

Erfahrungen: Refactoring

- KEIN Refactoring während des Projekts
- Gründe:
  - zu gründlicher Entwurf
  - Umschreiben des Programms ohne Testfälle
  - Kleine Projektgröße

Erfahrungen: Skalierbarkeit

- Mehr Teilnehmer beim Planungsspiel → mehr Kommunikation → größerer Zeitaufwand
- Lösen abstrakter Probleme besser in kleinen Teams
  - Geringerer Kommunikationsaufwand
  - Gruppendynamische Eigenschaften
- Teamgröße kritischer Faktor in XP
- Annahme: optimale Teamgröße liegt zwischen 6 und 8

Zusammenfassung

- Programmieren in Paaren schnell übernommen → wird als reizvolle Art der Softwareentwicklung empfunden
- Iterativer Entwurf in kleinen Schritten ist schwierig
- Test-first Entwicklung braucht Anlaufzeit, manchmal schwer zu verwirklichen
- XP definitiv nur für kleine Teams geeignet

Warum XP?
Methoden
Beispiele

- M. Sauer: *Extreme Programming - Grundsätze für Agile Software-Entwicklung,* Uni Erlangen, 2002
  http://www3.informatik.uni-erlangen.de/Lehre/UML-Seminar/SS2002/xprog.pdf
- D.Well: *Extreme Programming: A gentle introduction*
  http://www.extremeprogramming.org
- Hype Softwaretechnik GmbH
  http://www.hype.de
- M. Müller, W. Tichy: *Erfahrungen mit eXtreme Programming im universitären Umfeld*, ObjektSpektrum, Ausgabe 03/2001
- K. Beck: *Extreme Programming*, Addison-Wesley, 2000
- R. Jeffries, A. Anderson, C. Hendrickson: *Extreme Programming installed*, Addison Wesley, 2001
- M. Lippert, S. Roock, H. Wolf: *Software entwickeln mit XP*, dpunkt, 2002