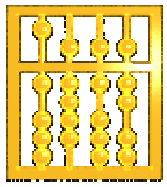
Extreme Programming (XP)

Überblick, Hintergründe, Perspektiven

Dr. Bernhard Rumpe
Software & Systems Engineering
Technische Universität München



Inhalt

- **Hintergründe zu XP**
- **Überblick über XP**
- **Technische und organisatorische Anteile in XP**
- **Kritische Betrachtung von XP**
- **Ausblick:**
  - **Extreme Modelling**
  - **Hierarchical XP**

XP ist

- ... nicht einfach „Hacking“,
- sondern eine „Light-weight“ Methode,
- bestehend aus zusammenpassenden „Best Practices“.

- Verzicht auf einige Elemente der Softwareentwicklung:
  - Dokumentation, Trennung in Phasen, ...
- Fokussierung stattdessen auf
  - Code, Tests, Kommunikation, ...
  - Sehr kurze Iterationszyklen, ...

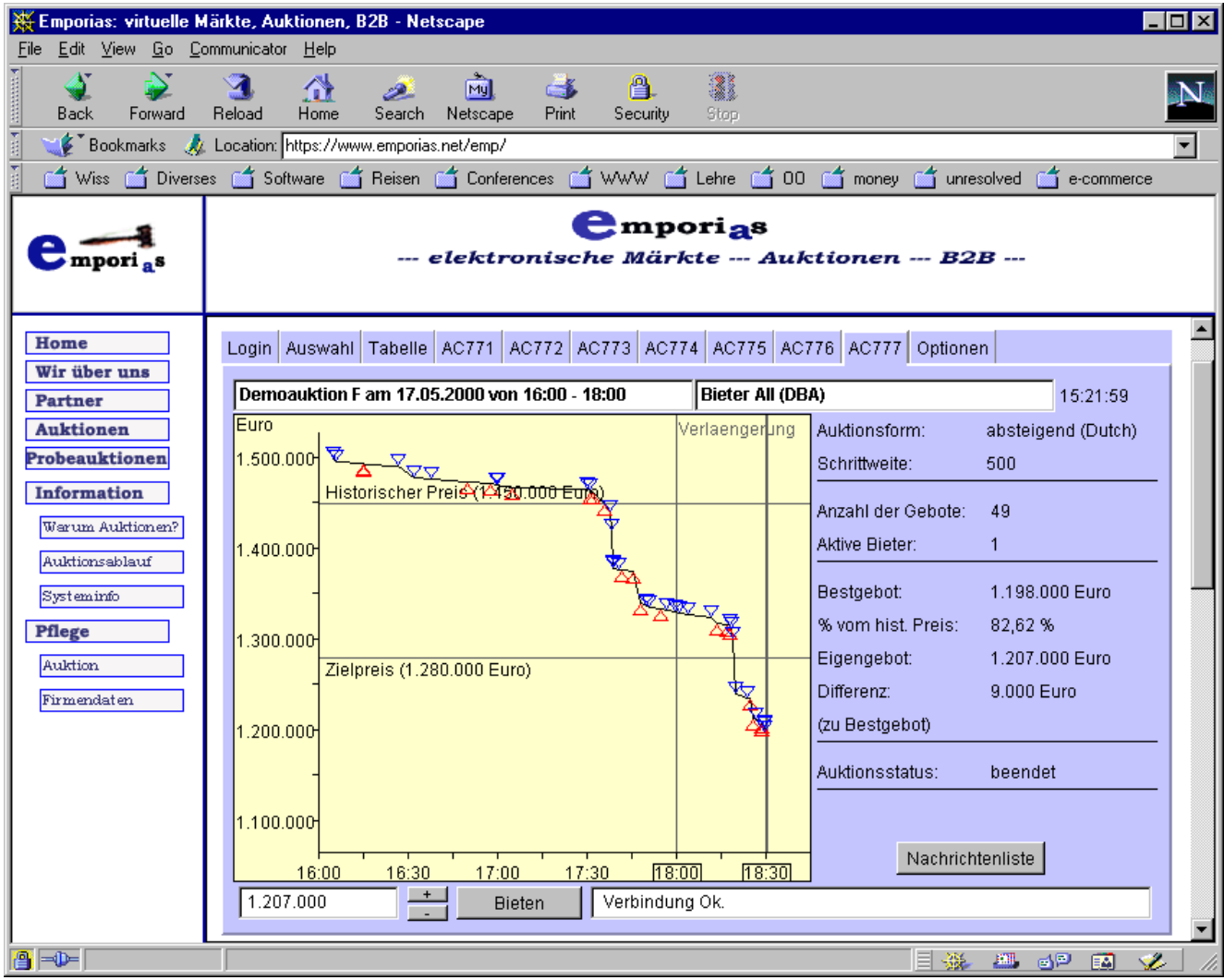
➪ XP kann nur auf bestimmte Projektarten angewendet werden.

Entstehung

- Kent Beck „Extreme Programming Explained“, Addison Wesley, 1999
  - einfach strukturiertes Buch, das wesentliche Elemente erklärt
- Basis von XP: Projekterfahrung bei
  - Schweizer Banken
  - Chrysler Payroll System (C3, ca. 1994/95)
- Erste Publikationen ab 1995, z.B. ECOOP´95
- Auf mehreren Web-Sites aktiver Erfahrungsaustausch
- XP als explizites Gegenstück zu RUP, Catalysis, UML, ...

XP an der TU München

- Beschäftigung mit XP seit Okt. 1999 (OOPSLA)
- Projekt in mehreren Phasen mit bis zu 10 Personen: E-Commerce/ Internet-Auktionen
- Betreuung weiterer Projekte bei Firmen



Motivation

- In kurzer Zeit ein qualitativ hochwertiges Software-System entwickeln
- Flexibel Anforderungen einbringen und priorisieren
- Programmierer konzentrieren sich frühzeitig auf wesentliche Elemente der Softwareentwicklung: Codierung.
- Hohe Innovationsgeschwindigkeit in der IT und sich ändernde Anforderungen machen flexibles Projektmanagement notwendig.

Rahmenbedingungen

- Maximal 5-10 Programmierer sind an einem Projekt beteiligt.
- Kunde steht zur intensiven Einbindung in das Projekt zur Verfügung.
- Räumlichkeiten erlauben intensive Kommunikation und Kontakt zwischen den Entwicklern.
- Kunde verzichtet auf ausführliche Dokumentation von Analyse und Entwurf.
- Entwickler-Team ist motiviert, eigenverantwortlich zu handeln.

Rollen in XP:

- Projektleiter: Der Projektleiter ist verantwortlich für Management und Koordination des Projekts. Er verwaltet Ressourcen, Kosten, Zeitpläne, um damit eine optimale Qualität zu erreichen.

- Kunde: Wenigstens ein Kunde ist permanent ansprechbar, um schnell aufkommende Fragen zu klären. Des weiteren entwirft der Kunde funktionale Tests für die Software.

- Entwickler: Entwickler tragen die Hauptlast der Projekts. Sie führen die vier Grundaktivitäten durch: Codierung, Testen, Zuhören, Design.

Aktivitäten (1)

Codierung

- In kleinen Schritten wird das System sukzessive erweitert
- Codierungsstandards
- Regelmäßige Durchläufe aller Tests
- Test werden gemeinsam mit dem Code entwickelt und fortgeschrieben.
- Zur Modifikation existenten Codes wird Refactoring eingesetzt.

Testen

- Jedes Programmelement besitzt einen automatisierten Test. Jede Schleife und jede Verzweigung werden dabei durchlaufen.
- Komponententests entstehen gemeinsam mit dem Code.
- Kunden entwerfen funktionale Tests, die die Geschäftslogik prüfen.

Aktivitäten (2)

Zuhören

- Kommunikation unter den Entwicklern, sowie zwischen Entwicklern und dem Kunden ist essentiell:
  - sie erlaubt flexibel die Erhebung und Weiterentwicklung von Anforderungen und
  - verhindert Irrwege bei der Entwicklung.

Design

- organisiert die Systemlogik in einer Form, die Änderungen lokal hält,
- hält Funktionalität und Daten zusammen,
- erlaubt die lokalisierte Erweiterung des Systems,
- ist schlank und verhindert unnötige Komplexität.
- Design ist Teil der täglichen Programmiertätigkeit.

Fundamentale Prinzipien

- **schnelles Feedback**
  - kontinuierliche Projektsteuerung
- **inkrementelle Änderungen**
  - keinen Big-Bang, sondern messbarer Fortschritt
- **qualitativ hochwertige Arbeit**
  - gesichert durch geeignete Massnahmen
- **Einfachheit**
  - Klarheit, Eleganz
- **Änderbarkeit unterstützen**
  - um Flexibilität zu erreichen
  - Fehlerkosten zu reduzieren

Entwicklungspraktiken

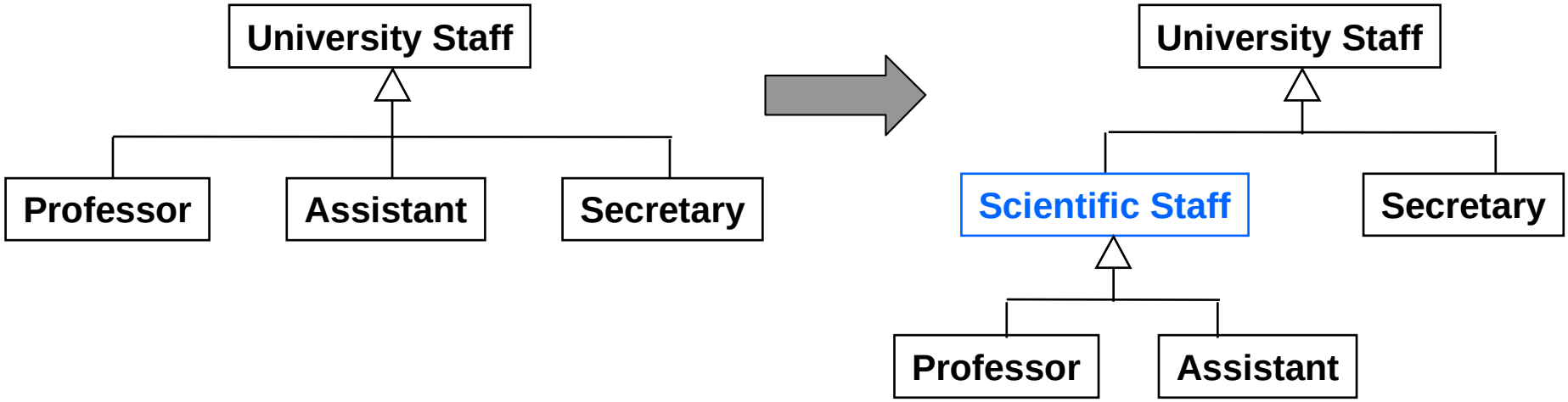
- **Planning game**
- Nutzung von Metaphern
- 40-Std. Woche
- ständig verfügbarer Kunde
- kleine Releases
- kontinuierliche Integration
- simples Design
- **Testen**
- **Refactoring**
- **Pair programming**
- gemeinsamer Code-Besitz
- Codierungsstandards

Planning Game

- ein evolutionärer, permanenter Dialog zwischen Notwendigem und Machbarem.
- Der Kunde entscheidet darüber, was zu implementieren ist:
  - Prioritäten
  - Zusammensetzung der Releases
  - Daten der Releases
- Entwickler entscheiden über:
  - Schätzungen Aufwand, Ressourcen
  - Konsequenzen
  - Vorgehensweise und Organisation

Refactoring

- Techniken zur inkrementellen Änderung des Programm-Codes.
  - Migration von Code entlang der Klassenhierarchie,
  - Zusammenlegung oder Teilung der Klassen.
- Sehr kleine Schritte, die systematisch angewendet werden

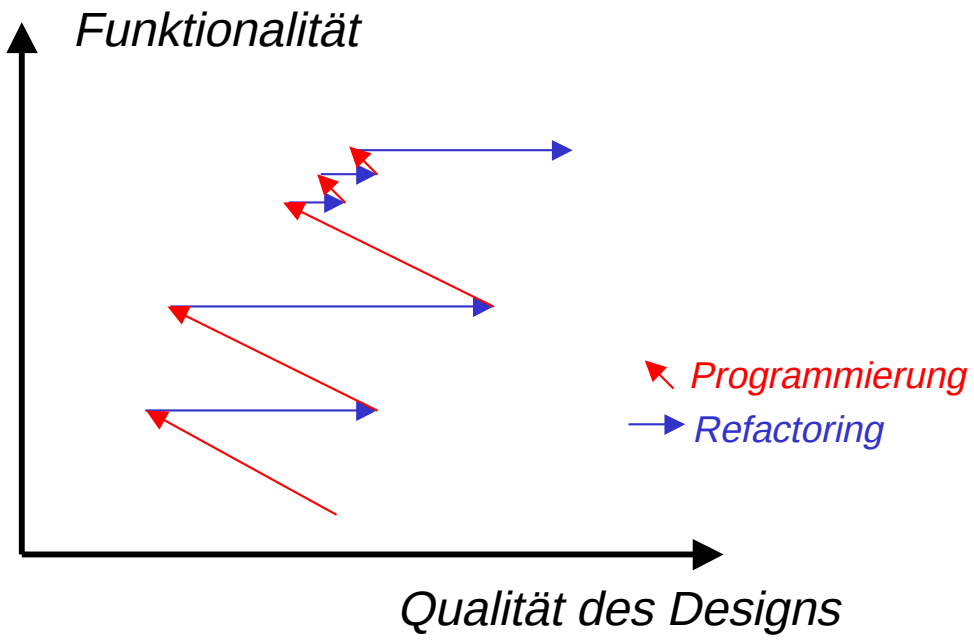


Einige Refactorings (Fowler,99)

- Add Parameter
- Collapse Hierarchy
- Encapsulate Collection
- Extract Interface
- Extract Method
- Move Field
- Move Method
- Pull Up Field
- Remove Middle Man
- Remove Parameter
- Rename Method
- Replace Array with Object
- Replace Conditional with Polymorphism
- Replace Delegation with Inheritance
- Replace Inheritance with Delegation
- Replace Error Code with Exception

Qualität des Designs

- Programmierung neuer Funktionalität und Refactoring sind ergänzende Tätigkeiten

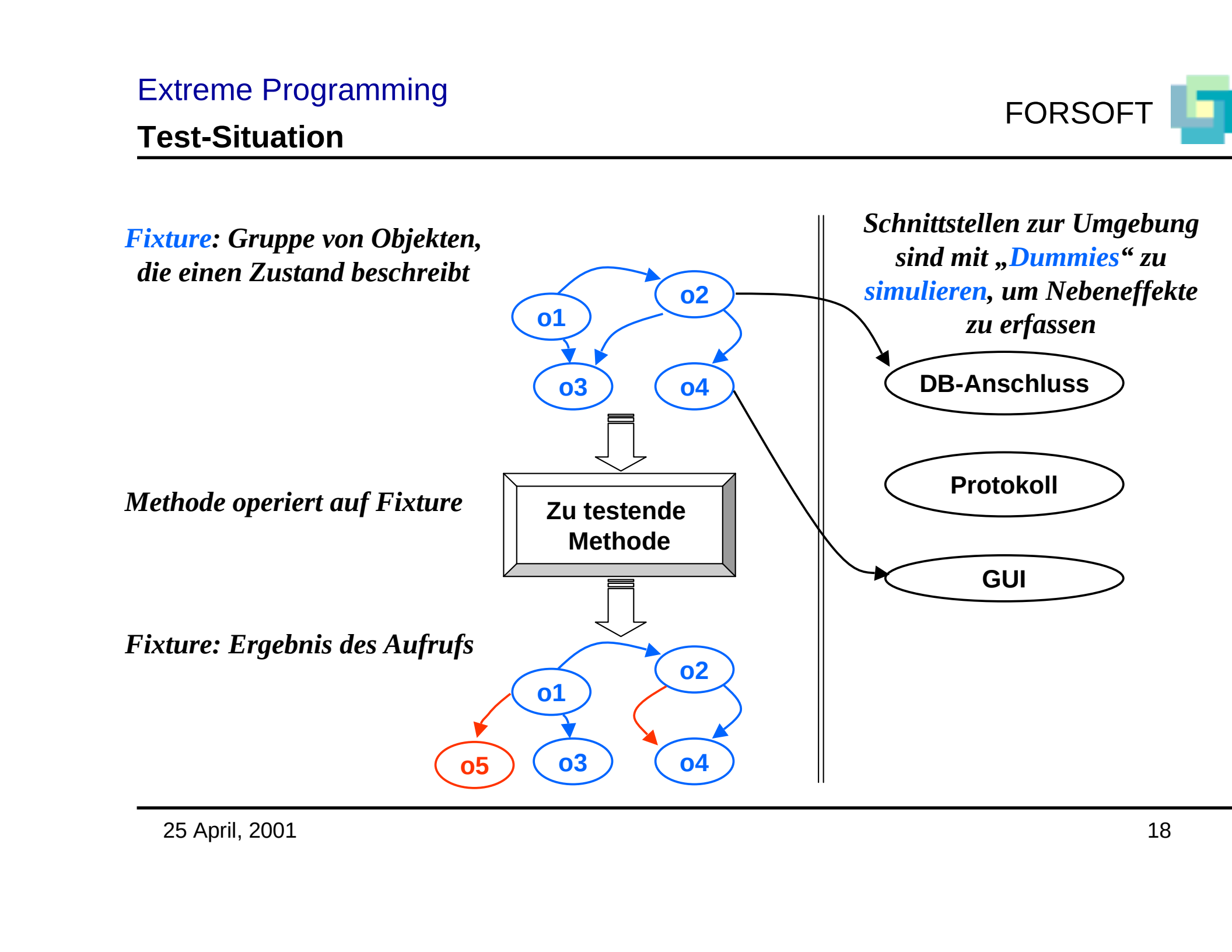


- geschickte Mischung erreicht Funktionalität und Qualität

Testen

- ist eine der wichtigsten Tätigkeiten in XP
- Parallele Entwicklung von Code und Tests garantiert „Testüberdeckung“
- Automatische Tests um Wiederholbarkeit zu sichern
- Tests als „Modell“ mit Prüfcharakter
- Tests sichern (weitestgehend) die Korrektheit von Refactorings
- Testarten:
  - Black-Box-Tests (entstehen bevor die Funktion geschrieben wird)
  - White-Box-Tests (decken die Kontrollflüsse ab)
  - Unit-Tests (vom Anwender entworfen)
- Framework *junit* unterstützt die Testdefinition

Test-Situation



Fixture: Gruppe von Objekten, die einen Zustand beschreibt

Methode operiert auf Fixture

Fixture: Ergebnis des Aufrufs

Schnittstellen zur Umgebung sind mit „Dummies“ zu simulieren, um Nebeneffekte zu erfassen

Paarweises Programmieren

- Vier Augen sehen mehr als zwei.
- Deshalb ist es in besonders kritischen Bereichen sinnvoll, dass zwei Personen gemeinsam an einem Terminal wesentliche Code-Stücke in Angriff nehmen.
- Zwei Personen an einem Rechner: eine tippt, die andere schaut über die Schulter – abwechselnd.

- Kritisch:
  - Muss zur Persönlichkeit der Beteiligten passen
  - Erste quantitative Versuche mit Studenten: (Paar vs. single)
    - Mehraufwand von ca. 30%
    - Time-to-finish: um ca. 15 % reduziert
    - Qualität (korrekte Testläufe): 97% pair, vs. 83% single

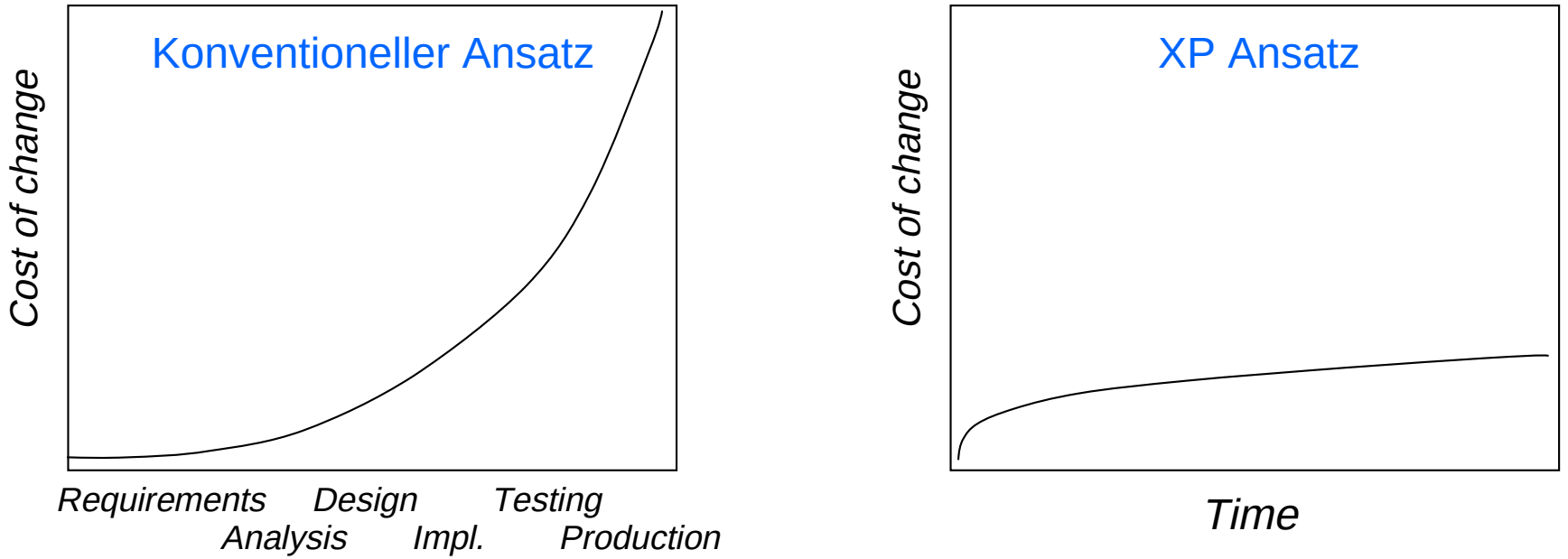
Pro & Contra

- Vorteile des XP:
  - Basiert auf „Best Practices“ und Entwickler-Erfahrung.
  - Betont die menschliche Komponente
    - z.B. 40h Woche, Kommunikation
    - Systematisiert „Hacking“ in einen Prozess
    - Kommt dem menschlichen Hacker-Naturell entgegen
  - Effizienz der Softwareentwicklung im Focus
- Probleme des XP:
  - Hohe Akzeptanzprobleme konventionell trainierter Entwickler
  - Skaliert nur bis max. 10 Personen
  - Manche Projekte schreiben Dokumentationsleistung vor
  - Anwendung auf manche Einsatzgebiete unklar, z.B. Embedded Systems

Wissenschaftliche Aufarbeitung von XP?

- XP basiert auf einigen Annahmen, die nicht validiert sind:
  - Viele Änderungswünsche der Kunden
  - Einarbeitung in dokumentierten Code ist wenig aufwendig
  - Lineare Kostenkurve bei Fehlerbehebung
- „Best Practices“ sind auf ihre Wirkung nicht ausreichend untersucht:
  - Pair Programming
- Objektive Berichterstattung über XP existiert praktisch nicht.
- Konzepte, die XP unterstützen können, werden nicht eingesetzt:
  - Testmethodik
  - Refactoring-Werkzeuge

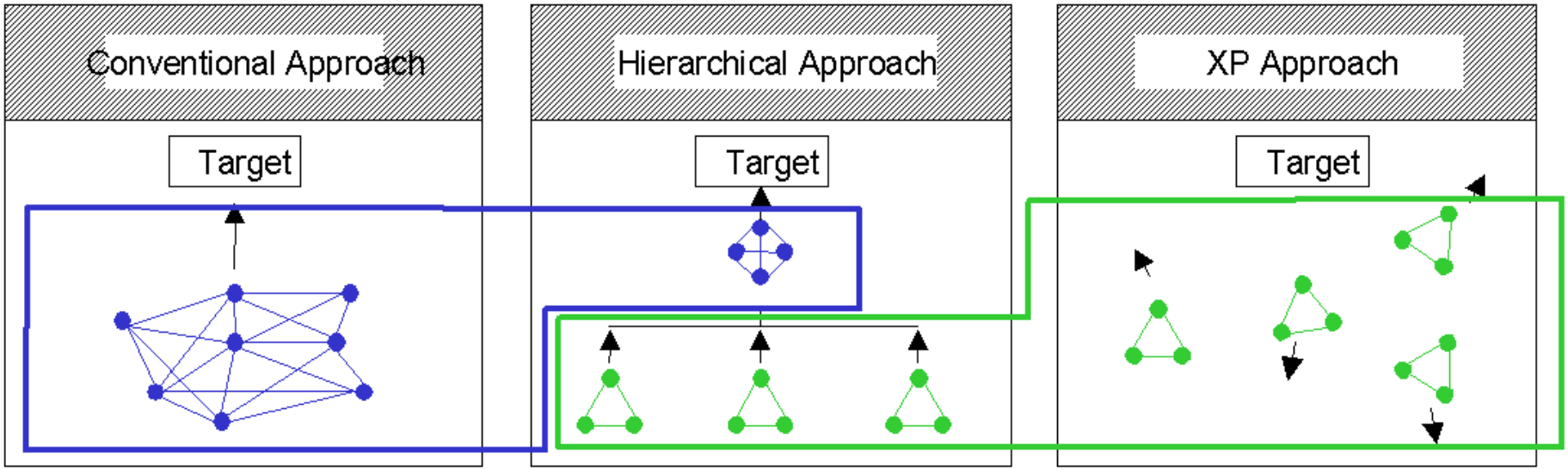
Lineare Kostenkurve bei Fehlerbehebung?



- „cost of change may not rise dramatically over time“
- Dies ist eine wesentliche Annahme von XP, obwohl diese Annahme nicht empirisch belegt ist.

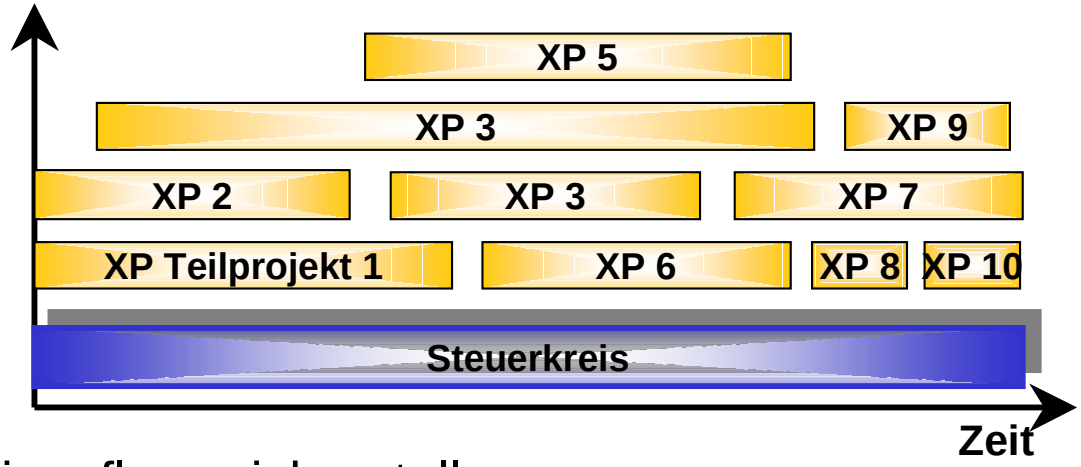
Exkurs: Hierarchical XP

- **Grundidee:**
  - **ein großes Projekt = viele kleine XP Projekte + Steuerkreis**



Steuerkreis

- Rollen:
  - Ein Mitglied jedes Teilprojekts
  - Ein Werkzeugverantwortlicher, Versionsmanager, …
  - Projektleiter
  - mindestens ein Kunde



- Tasks:
  - Kommunikation, Informationsfluss sicherstellen
  - Schnittstellen verwalten und entwickeln
  - Teilprojekte dynamisch arrangieren
  - Technologiefragen klären
  - Risikomanagement

Exkurs: Extreme Modelling

- Neuere Entwicklung mit folgender Überlegung:
  - Verwende Techniken des XP und wende sie auf „executable UML“ statt Java an.
- Einige Werkzeughersteller entwickeln derzeit auf UML Basis:
  - Code Generatoren
  - Test Generatoren
  - Refactoring Browsern
- Ziele:
  - Noch kürzere Entwicklungs-Zyklen,
  - kompakteren „Code“,
  - kommunizierbare Modelle für Entwurf und Implementierung
  - noch schnellere Entwicklung.

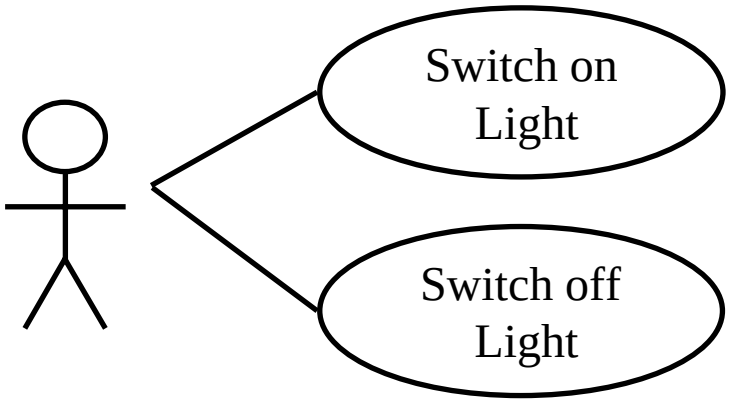
Xtreme Modelling als Synthese

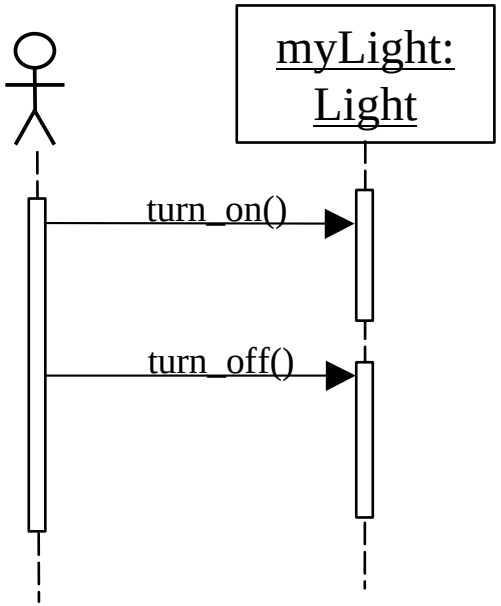
- Durch die Anwendung von Techniken des XP auf die Modellierung kann ein Vorgehensmodell entstehen, das die Vorteile von XP und mit den Vorteilen von Modellierungstechniken verbindet.

- Voraussetzung sind auch gute Werkzeuge, die Modelle ausführen und Produktions- sowie Test-Code generieren können.

- Z.B. baut die Firma Gentleware Werkzeuge zum Entwerfen, Ausführen und Testen von Modellen und zur Code-Generierung

Ein einfaches Beispiel: Lichtschalter

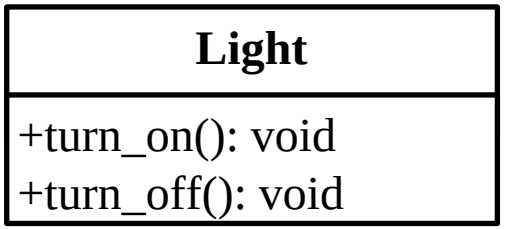




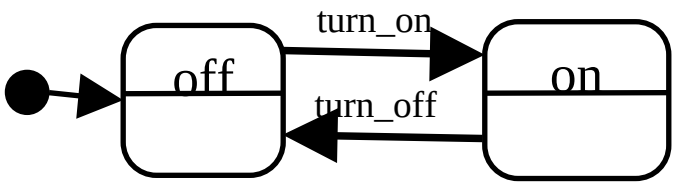
- Zunächst werden Use-Cases/User Stories identifiziert
- Diese werden durch Sequenzdiagramme erläutert (Test-First-Modeling)
- Aus den Sequenzdiagrammen wird später Testcode generiert

Ein einfaches Beispiel: Lichtschalter II

- Dann werden Klassen und Schnittstellen entwickelt, so dass die Tests alle compilieren (vom Modellierungtool gesteuert)

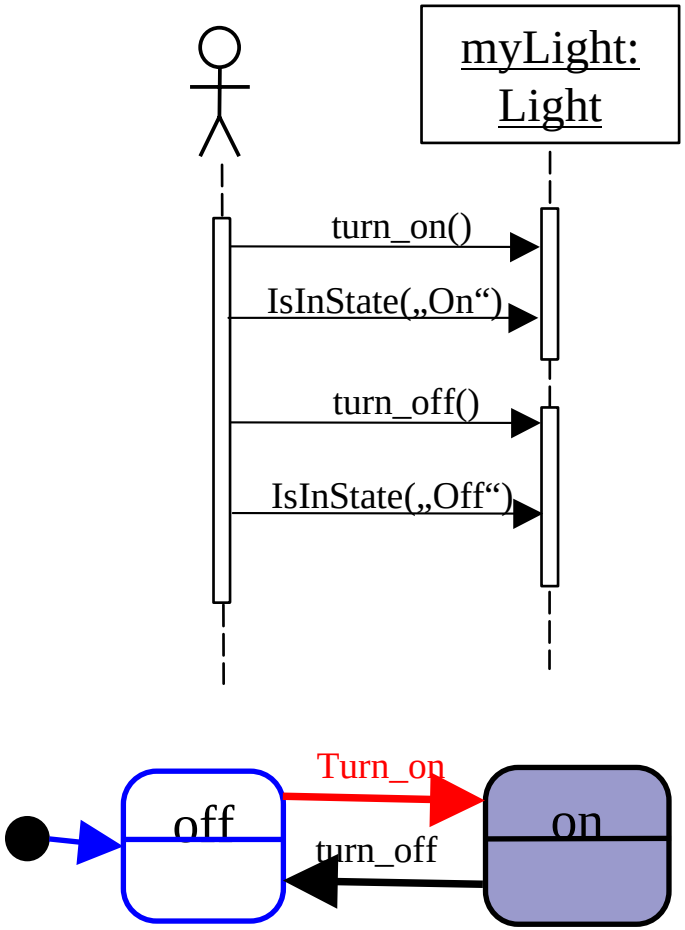


- Beschreibung des Verhaltens durch Zustandsdiagramme



- Code-Generierung aus Klassen- und Zustandsdiagrammen

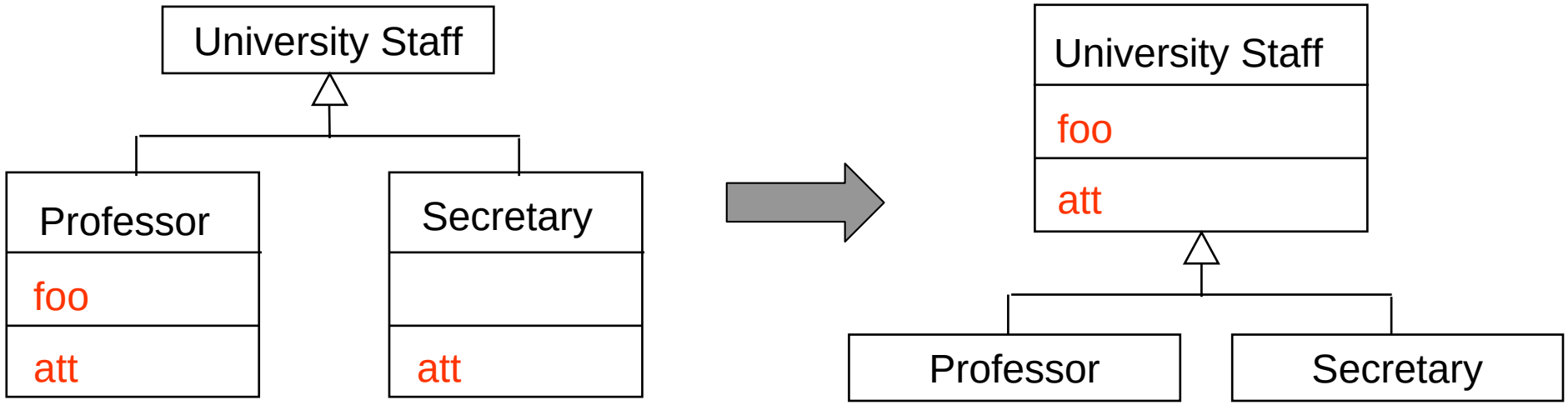
Ein einfaches Beispiel: Lichtschalter III



- Nun können die Tests ausgeführt werden und das Erreichen der Zustände visuell oder automatisch überprüft werden
- Anschluss zum Modelchecking
- Schließlich werden algorithmische Anteile nach XP entwickelt.
- Tests und Code werden weiterverwendet.

Beispiel: Refactoring in Klassen Diagrammen

- Verschieben von Features in einer Generalisierungshierarchie auf Knopfdruck



Auswirkungen des Extreme Modelling?

- Welche Effekte hat die Ausführbarkeit einer Modellierungssprache:
  - sie verkommt zur Programmiersprache:
  - verleitet „Programmierer“ Details auszufüllen, statt das „Big Picture“ zu modellieren
  - Effizienz ist wichtiger als Klarheit des Modells
  - Operationale statt eigenschaftsorientierte Modell

  + Very Rapid Prototyping
  + Keine Redundanz (Inkonsistenz) zwischen Modell und Implementierung

Resümee

- XP ist kein Teufel oder kein heiliger Gral
- XP besitzt Potential, Softwareentwicklung in bestimmten Bereichen quantitativ und qualitativ weiter zu verbessern
- XP gehört ins Portfolio des Softwareentwicklers

- Fundierte Untersuchungen der Effekte von XP fehlen
- Integration von weiteren Konzepten in XP hat noch stattzufinden:
  - Testmethodik
  - Verifikations- und Validierungstechniken

➪ Ziel: Weiterentwicklung von XP in Richtung Modellierungstechniken.