Vorlesung Software Engineering

Foliensatz Nr. 13 (12.01.12)

Arbeitsgruppe Software Engineering
Prof. Elke Pulvermüller

Universität Osnabrück
Institut für Informatik, Fachbereich Mathematik / Informatik
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

elke.pulvermueller@informatik.uni-osnabrueck.de

http://www.inf.uos.de/se

Sprechstunde: mittwochs 14 – 15 und n.V.

Inhalt

Inhalt

6.7 Microsoft Vorgehen

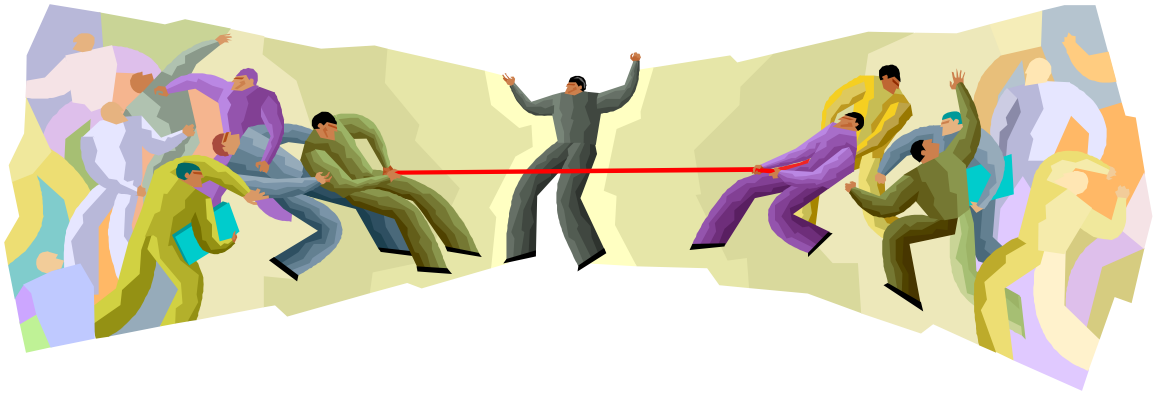
Besonderheiten:

- Kein Auftraggeber sondern Konfektionsware (“shrink-wrap software”)
  - Antizipation der Interessen großer Kundenkreise
  - keine typischen Analysemethoden
- Hohe Flexibilität und Ausrichtung auf unternehmerisches Risiko
  - keine langen Produktzyklen oder große Verwaltungsstrukturen
  - “Hackerkultur” (Was ist ein Hacker?)

[CS97] M. Cusumano, R. Selby: How Microsoft Build Software. Communications of the ACM, 40(6):53 – 61, 1997

6.7 Microsoft Vorgehen

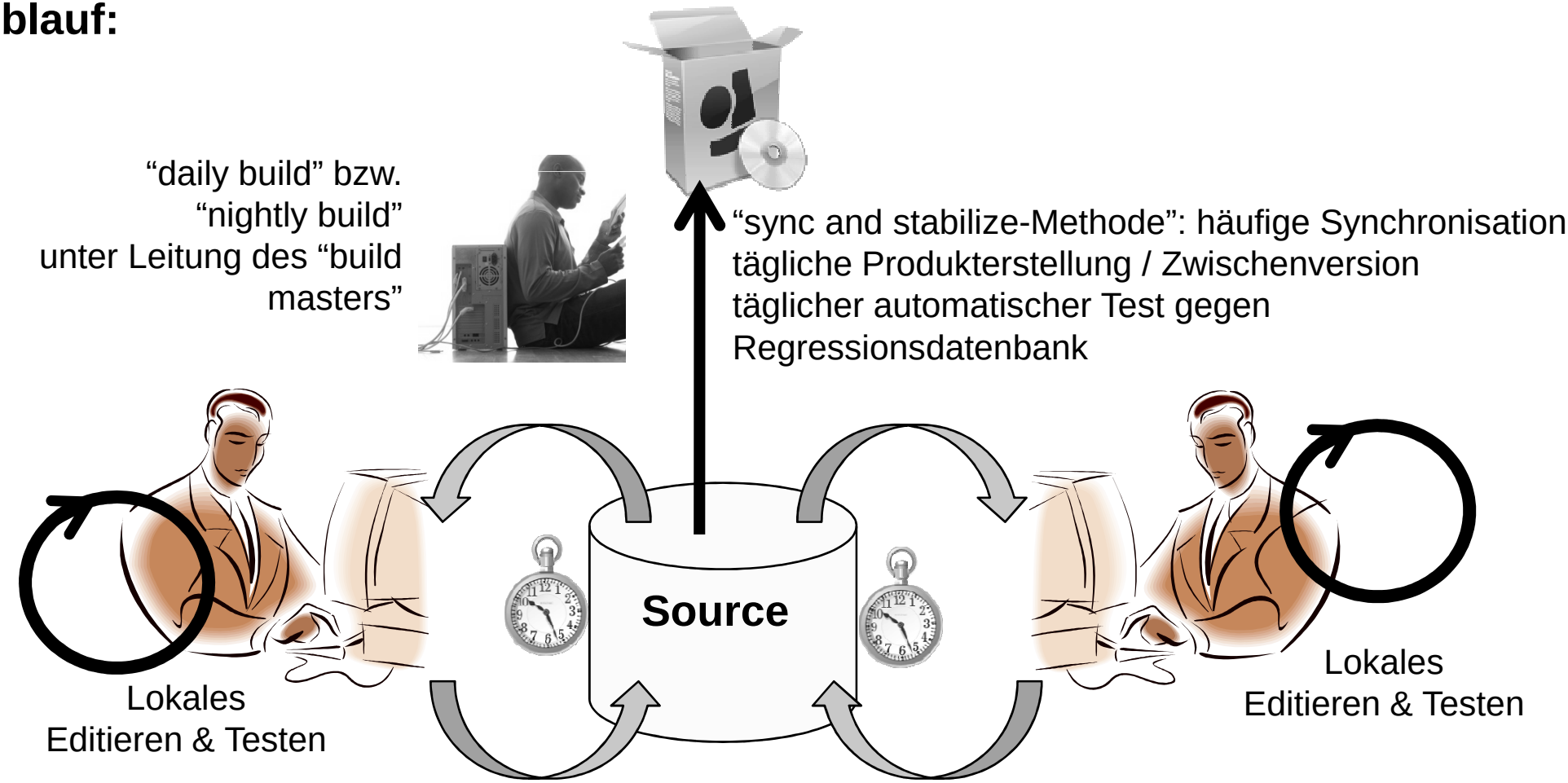
Organisation in flacher Hierarchie:



- lose strukturierte kleine Entwicklerteams (3 – 8) Entwickler arbeiten parallel
- etwa gleich große Gruppe von Softwaretestern begleitend pro Entwicklerteam
- Feature-Orientierung mit Gesamtverantwortung und Autonomie

6.7 Microsoft Vorgehen

Ablauf:



- Code zurück in die zentrale Datenbank: Mindestens 2x pro Woche
- Code muß compilierbar und bindbar sein (keine Fehler im fertigen Produkt!)

6.7 Microsoft Vorgehen

Aufwendig aber positiv für die Projektfortschrittskontrolle:

“Wir bauen jeden Tag, und zwar genau um 17 Uhr, eine Zwischenversion [...] selbst wenn am nächsten Tag Feiertag ist, und wir genau wissen, daß niemand die Zwischenversion benötigt. Alle Beteiligten bekommen ein Gefühl für den Prozeß und sehen, daß das Projekt kontrolliert abläuft.”

“Für uns ist wichtig, daß alle Beteiligten vor Feierabend synchronisieren, daß also jeder Entwickler seine Änderungen abgleicht. Wenn man dies aufschiebt, bekommt man irgendwann einmal größere Konflikte beim Zusammenmischen.”

“Wir hatten damals Ziegenhörner. Wer einen Abbruch verursacht, mußte die Ziegenhörner aufsetzen und so lange aufbehalten, bis ein anderer den Zwischenversionsbau zum Absturz brachte.”

[CS95] M. Cusumano, R. Selby: Microsoft Secrets: How the World´s Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People. The Free Press, Simon & Schuster, New York, 1995

6.8 Agile Modelle

- **Agile Modelle =** **eine Menge alternativer Software-Entwicklungsprozesse, die auf gemeinsamen Prinzipien basieren**
- **Motivation:**

Traditionelle, disziplinierte Prozesse ⟺ **"Code-and-Fix"**

(diszipliniert, detailliert, planbasiert, dokumentenorientiert, bürokratisch, langsam, häufig von Entwicklern abgelehnt)

→ **Agile Software-Entwicklungsprozesse „Agile Manifesto"**

[AgileAlliance 01: http://agilemanifesto.org/]

- **Paradigma:**

Vorhersagbarkeit (predictability)	Anpassbarkeit (adaptability)	Schnelle Umsetzung

- **Einsatz:**

Große, komplexe und lang andauernde Projekte	Kleinere Projekte mit unklaren und wechselnden Anforderungen (**?**)	„Hello World"

6.8 Agile Modelle

- **Das „Agile Manifesto“:** [AgileAlliance01]

„Individuals and interactions	over	processes and tools
Working software	over	comprehensive documentation
Customer collaboration	over	contract negotiation
Responding to change	over	following a plan“

- ⇒ **Entwicklungsstil auf der Basis eines Wertegerüsts:**

[AgileAlliance01] Agile Alliance (Jim Highsmith, Kent Beck, Ward Cunningham, Martin Fowler, Steve Mellor, ...), http://agilemanifesto.org/

6.8 Agile Modelle

Beispiele für konkrete agile Software-Entwicklungsprozesse

Adaptative Software Development (ASD, Highsmith 00)

Feature Driven Development (FDD, Coad 99)

Crystal Methoden (Alistair Cockburn 01)

Dynamic Software Development Method (DSDM)

Rapid Application Development (RAD)

Scrum (oder xP@Scrum), Mitte 90er

(E)Xtreme Programming (XP)

Agile Unify Process (AUP)

Lean Development (LSD, Poppendieck 01)

Für alle gilt:
Menschen, Kunden,
Kommunikation,
laufende Software,
Reaktion auf Änderungen
sind die zentralen Faktoren

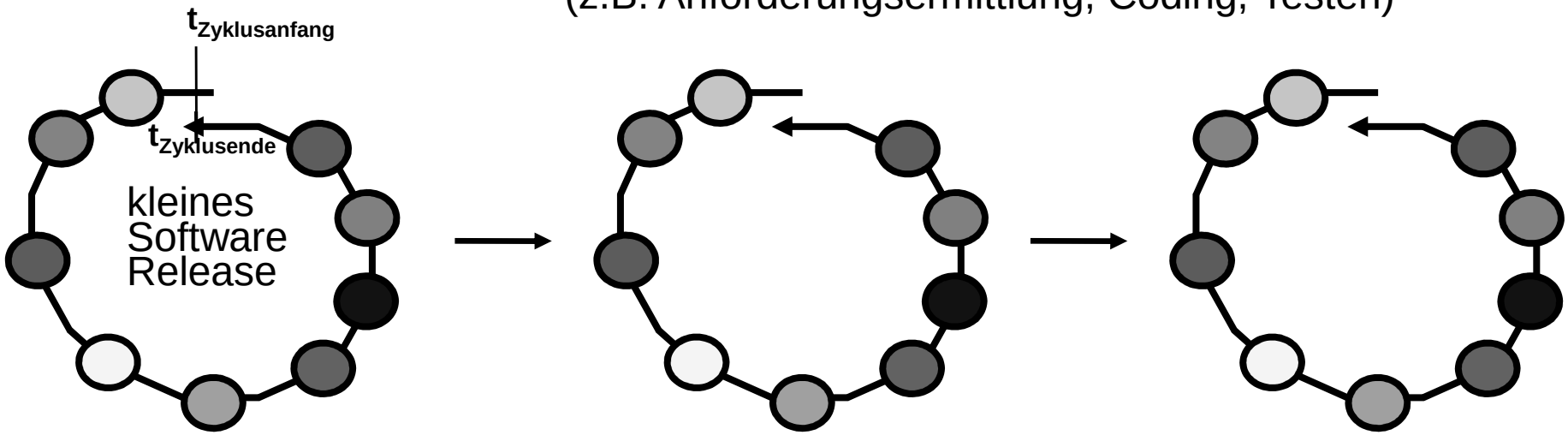
- **Weiterführende Literatur und Vergleich der Prozesse z.B. in:**
  Jim Highsmith, Agile Software Development Ecosystems, Addison Wesley, 2002

6.8 Agile Modelle

Entwicklungsprinzipien agiler Software-Entwicklungsprozesse zur Erreichung der aufgestellten Werte:

- Iterative Arbeitsabläufe (Workflows)
- Inkrementelle Auslieferung lauffähiger Software
- Kurze Iterationen fixer Länge (“time-boxed”)

  $\Rightarrow$ Zykluszeit: festgelegte Zeit pro Iteration
  (Wochen bis zu 3 Monaten)
- Viele parallele Aktivitäten innerhalb einer Iteration
  (z.B. Anforderungsermittlung, Coding, Testen)



pro Iteration: eine Auslieferung eines kleinen lauffähigen Teilsystems

verschiedene Grade an Zermonie („ceremony“): #Artefakte (Arbeitsergebnisse), #Schritte

6.8 Agile Modelle

Best Practices bei agiler Softwareentwicklung

- Entwickle iterativ
- Modelliere graphisch
- Überwache Erfüllung der Anforderungen
- Verfolge Änderungen
- Schließe jeden Schritt mit Qualitätsprüfung ab
- Verwende eine komponentenbasierte Version
- Entwickle nur was zur Lösung notwendig
- Konzentriere Dich auf die wesentlichen Ergebnisse und weniger darauf, wie sie erzielt werden
- Vermeide unnötige Dokumente
- Passe Dich an den Entwicklungsstand an
- Lerne von Fehlern
- Überprüfe regelmäßig die Risiken des Projektes
- Entwickle objektivierbare Kriterien zur Messung des Projektfortschrittes
- Versuche Routinearbeit zu automatisieren
- Arbeite nach Plan

6.8 Agile Modelle: Extreme Programming

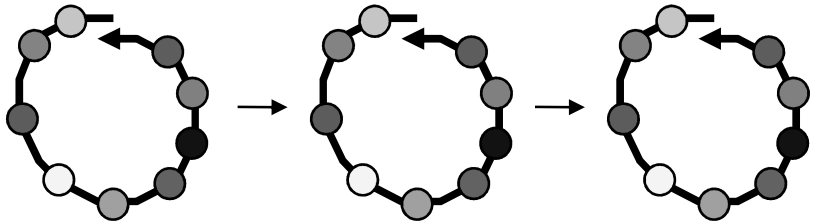
UNIVERSITÄT OSNABRÜCK

Beispiel eines verbreiteten agilen Modells: Xtreme Programming / Extreme Programming (XP)

- **Eingeführt ca. 98/99**
- **Ausgangsfragestellung:**

  „Was würde passieren, wenn wir jede Technik / Praxis hernehmen würden und sie bis ins Extreme ausführen würden? Wie würde dies den Software Prozess beeinflussen?“

  ⇒ **hohe Anforderungen an Disziplin, Teamarbeit und Fertigkeiten**



- Programmiererzentrierte Methode
- Betonung technischer Praxis
- sehr kurze Iterationen (1 – 4 Wochen)
- niedriger Grad an Zeremonie (wenige Artefakte: „Story Cards“, Code, Tests)

⇒ **Ableitung von 15 Prinzipien und und 13 (sich gegenseitig stützenden) Praktiken**

6.8 Agile Modelle: Extreme Programming

4 spezielle Kernwerte (values):

Einfachheit (simplicity),

Mut (courage)

Kommunikation (communication)

Kundenrückmeldung (feedback)

15 XP Prinzipien (basic principles):

Unmittelbares Feedback (Rapid Feedback)

Einfachheit anstreben (Assume Simplicty)

Inkrementelle Veränderung (Incremental Change)

Veränderung wollen (Embracing Change)

Qualitätsarbeit (Quality Work)

6.8 Agile Modelle: Extreme Programming

4 Kernwerte:

Lernen lehren (Teach Learning)

Die weiteren XP Prinzipien:

Geringe Anfangsinvestition (Small Initial Investment)

Auf Sieg spielen (Play to win)

Gezielte Experimente (concrete Experiments)

Offene, aufrichtige Kommunikation (Open, honest Communication)

Die Instinkte des Teams nutzen, nicht dagegen arbeiten (Work with people´s instincts, not against them)

Verantwortung übernehmen (Accept Responsibility)

An örtlichen Gegebenheiten anpassen (Local Adaptations)

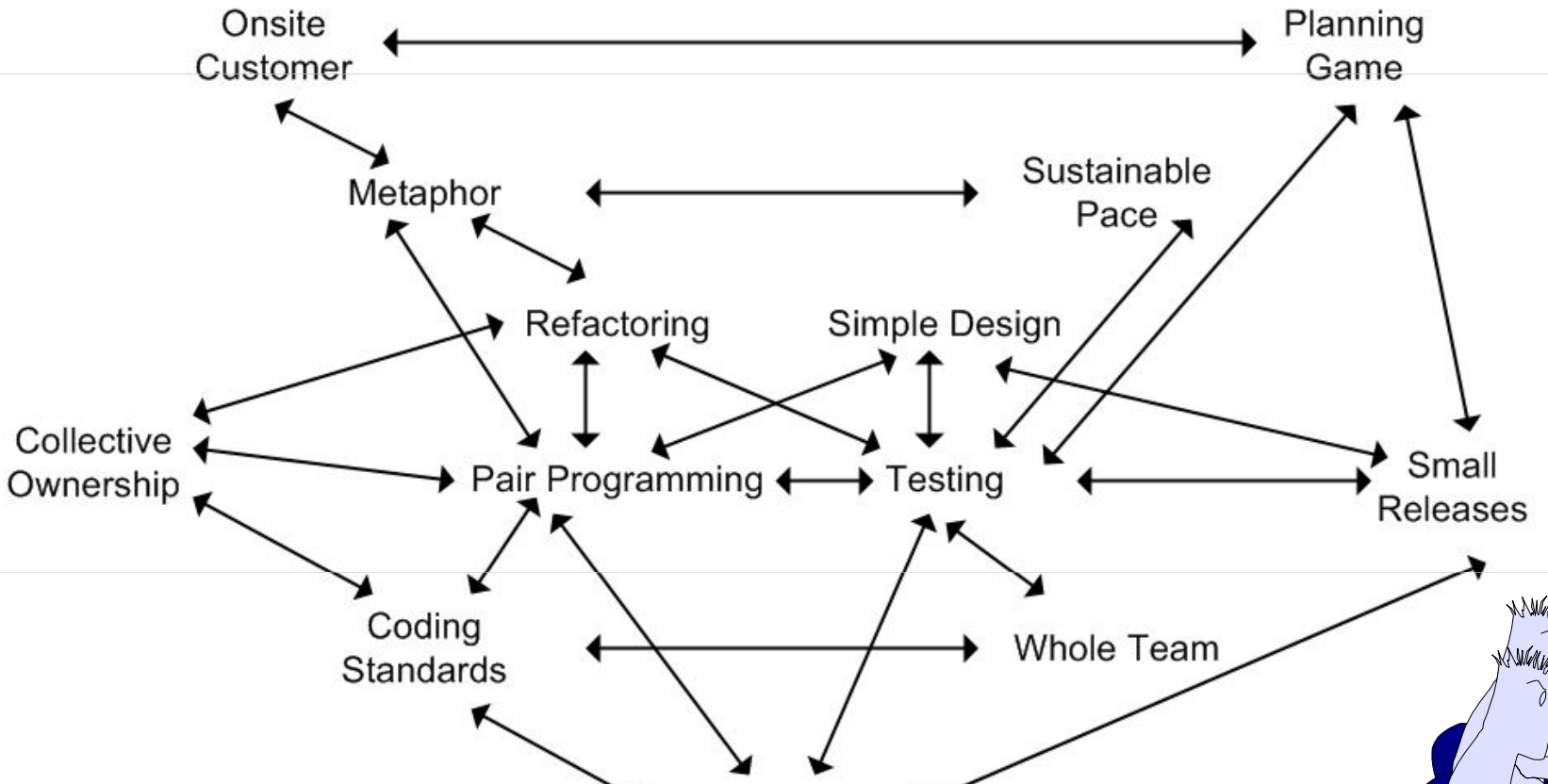
Mit leichtem Gepäck reisen (Travel light)

Ehrliches Messen (Honest Measurements)

6.8 Agile Modelle: Extreme Programming

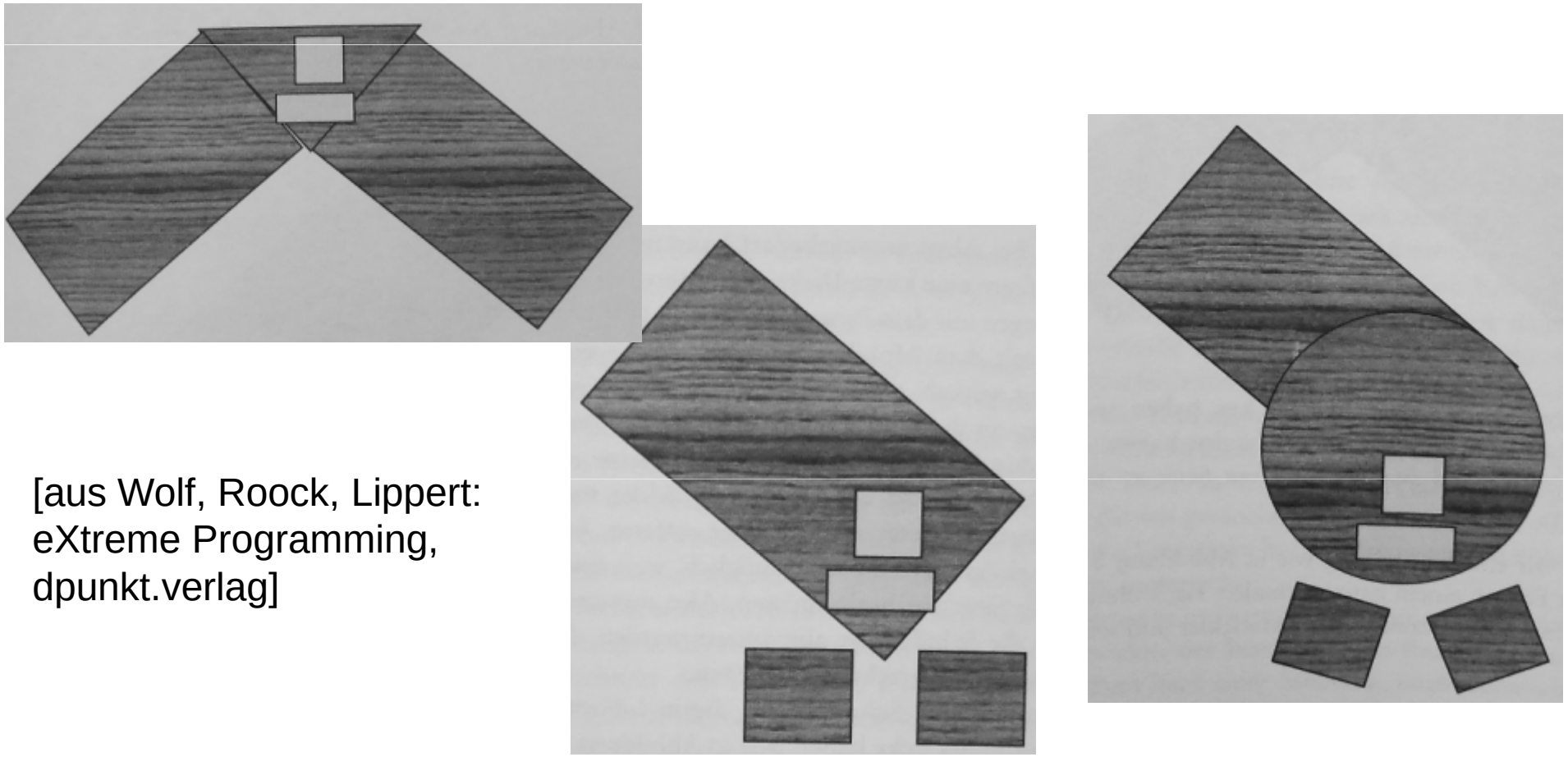
UNIVERSITÄT OSNABRÜCK

13 Praktiken (practices):



Stand-Up Meeting
Retrospection

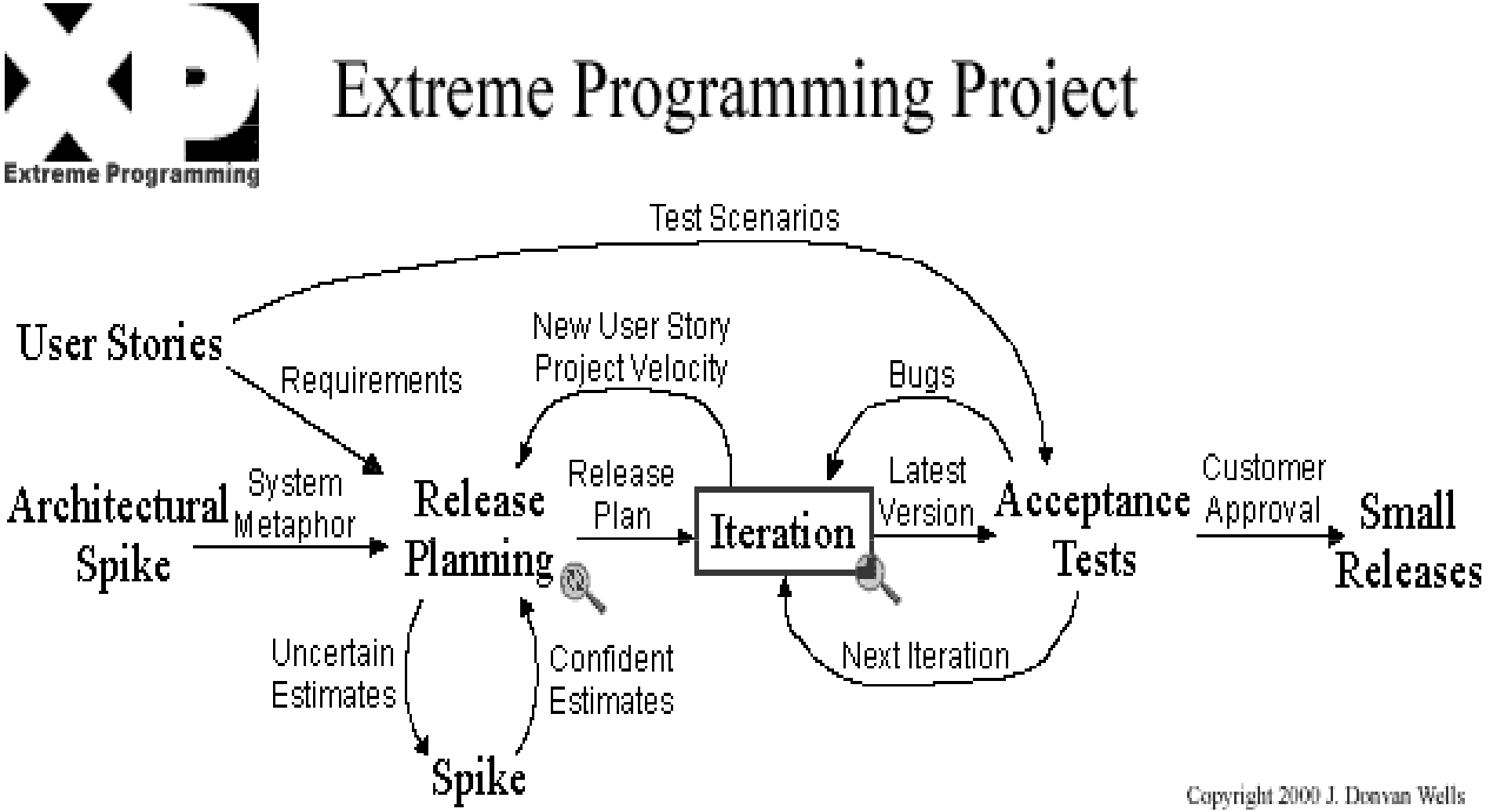
13 Praktiken (practices): Pair Programming



[aus Wolf, Roock, Lippert: eXtreme Programming, dpunkt.verlag]

6.8 Agile Modelle: Extreme Programming

Gesamtbild



6.8 Agile Modelle: Extreme Programming

Start: Ermittlung der Benutzeranforderungen durch „Stories“:

„Stories“ = Features, die ein Kunde will

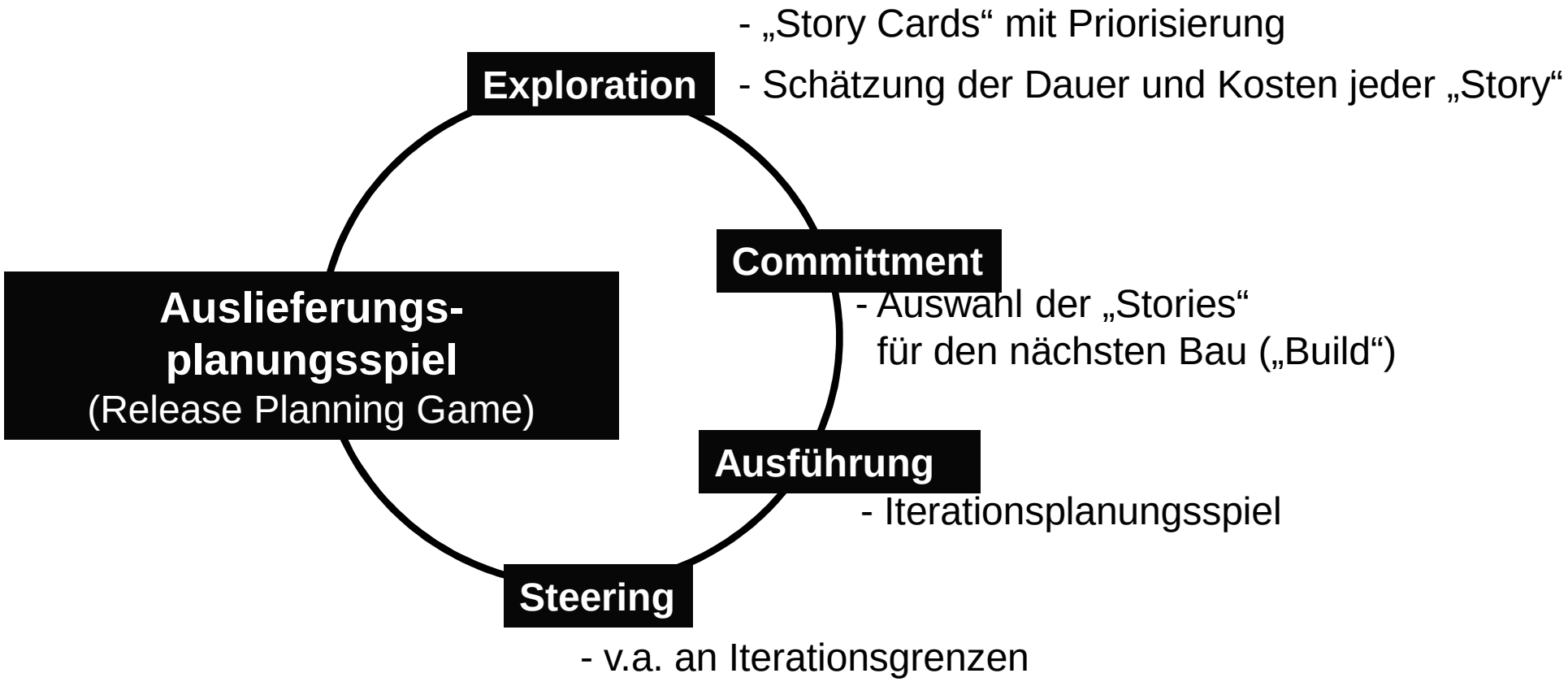
Notizkarten als Beschreibung und Erinnerung der versprochenen Funktionalität

Konversation zwischen Entwickler und Anforderungsgeber

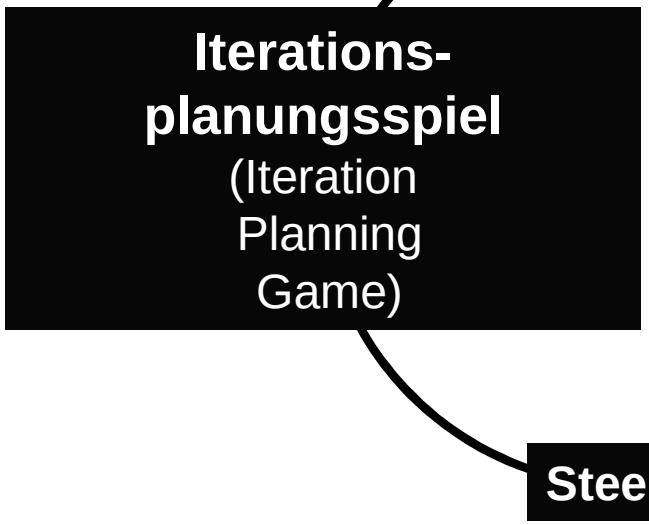
Tests (aller Art: Unit, Integration, Akzeptanz, etc.)

- Schätzung der Dauer und Kosten jeder „Story“
- Auswahl der „Stories“ für den nächsten Bau („Build“)
- Jeder Bau ist in „Tasks“ (Ziele) unterteilt
- Testfälle werden für eine Task zuerst notiert (Test-driven Development)

Planungsaktivitäten: Auslieferungsplanung, Iterationsplanung



Planungsaktivitäten: Auslieferungsplanung, Iterationsplanung



- Unterteilung von „Story Cards“ in Tasks
- Schätzung der Dauer und Kosten der Tasks

Exploration

Committment

- Übernahme der Verantwortung für Tasks

Ausführung

- „Partnering“ (Pair Programming)
- Formulierung von Testfällen für die Tasks
- Codierung der Tasks
- Komponenten- / Akzeptanztests

Steering

Beck, K.:
Extreme Programming explained,
Addison Wesley, 2000

6.8 Agile Modelle: Extreme Programming

UNIVERSITÄT OSNABRÜCK

Rollen in XP Projekten:

Kunde (customer):	- erzeugt und priorisiert „Stories“ - kontrolliert Releases (Umfang, Auslieferungsdatum)
Programmierer:	- gemeinsames Schätzen der „Stories“ - individuelles Schätzen und Entscheidung zur verantwortlichen Übernahme von Tasks - erarbeiten von Testfällen - implementieren
Coach:	- überwachen (monitoring) des Software-Entwicklungsprozesses - Team Mentoring (XP Prozess und Technik) - Aufmerksammachen auf potentielle Probleme und Optimierungen
Tracker:	- überwachen (monitoring) des Fortschritts - alarmieren, falls Planungsanpassungen und Rebalancierung von Tasks notwendig erscheint

Bewertung

- Achtung: agil (bzw. XP) heißt nicht chaotisch oder anarchisch, sondern erfordert im Gegenteil viel Disziplin (z.B. “Verantwortung für den Code”)
- Oft vordergründige Einfachheit der Konzepte (z.B. Metapher auch als Architekturvorgabe, Einfachheit), einfach verständlich, schwer in der Durchführung
- Problematische Vertragsgestaltung
- Reale Umsetzung sieht oft anders aus (z.B. bei Besprechungen setzt man sich doch ...)
- Eher für kleinere Projekte (5 – 15 Personen)
- Projektspezifische Auswahl (jedes Projekt braucht eine spezifische Menge von Policies und Konventionen) und schrittweise Einführung der Praktiken (z.B. alle 2-3 Wochen ein Set)
- Größere Projekte: eher Scrum, Crystal, FDD

6.9 Weitere Phasenmodelle

- **… gibt es viele**
- **Alternative Vorgehenspläne für ingenieursmäßiges Vorgehen zur Software-Entwicklung (Phasen, Anordnung der Phasen und Wirkung)**
- **Kriterien zur Entscheidung für einen bestimmten Vorgehensplan umfassen u.a. :**
  - das Unternehmen
  - dessen Management
  - die Fähigkeiten der Angestellten
  - das Projekt und die Natur des Produkts
- **Im konkreten Einsatz häufig:**
  - „Mix-and-match“ Entwicklungsmodell und “Tailoring”

6.10 Fokus: Analysephase, Requirements Engineering

- **In der Anforderungsphase werden durch Klärung und Präzisierung der Kundenwünsche die Anforderungen an das zu entwickelnde System festgelegt („Was“)**

- **Requirements Engineering = Anforderungsfestlegung, Aufgabendefinition, …**

Das *Requirements Engineering* beschäftigt sich
mit der systematischen, ingenieurmäßigen Entwicklung
einer Anforderungsdefinition, welche die Leistungen eines Systems
vollständig und eindeutig beschreibt.

- **Als Ergebnis des Requirement Engineering entsteht ein Anforderungsdokument (Pflichtenheft, Lastenheft, Produktdefinition, Requirementsspezifikation).**

6.10 Fokus: Analysephase, Requirements Engineering

- **Inhalt des Anforderungsdokuments:**

  - Funktionale Anforderungen (welche Funktionen soll das System erfüllen?, Ein-/Ausgabeverhalten jeder Funktion?)
  - nicht-funktionale Anforderungen (z.B. Speicherbedarf, Dokumentation, Richtlinien, physikalische Betriebsbedingungen)

- **Aufgabe des Anforderungsdokuments:**

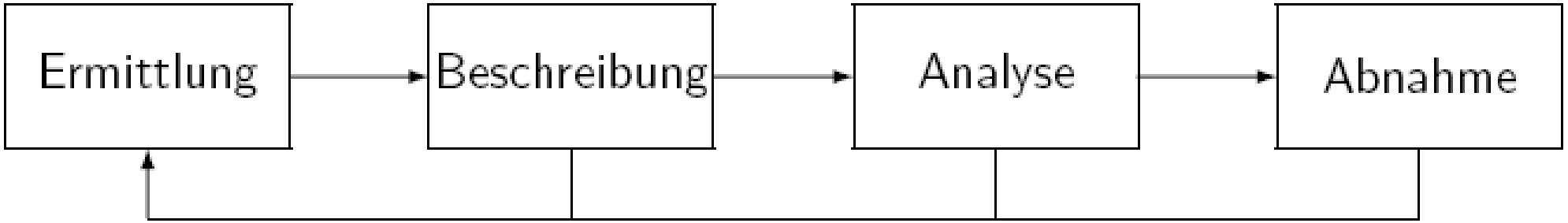
  - Kontrakt (Auftraggeber, Auftragnehmer)
  - Validierung (Benutzerwünsche)
  - Ausgangspunkt für Spezifikation/Grobentwurf

6.10 Fokus: Analysephase, Requirements Engineering

- **Tätigkeiten im Requirements Engineering:**

Problemanalyse

Anforderungs-analyse (Vollständigkeit, Konsistenz, Eindeutigkeit, …)



Anforderungsdefinition (gruppieren, klassifizieren, …)

6.10 Fokus: Analysephase, Requirements Engineering

UNIVERSITÄT OSNABRÜCK

- **Mängel und Fehler in Anforderungsdokumenten**

  - fehlende, dürftige, mehrdeutige Information
  - überholte, irrelevante Information
  - falsche, redundante, schlecht überprüfbare, unklare Anforderungen
  - undurchführbare, widersprüchliche, überspezifizierte Anforderungen

- **Weitere Aspekte des Requirements Engineering**

  - geeignete Beschreibungsmittel
  - methodisches Vorgehen
  - unterstützende Werkzeuge

6.10 Fokus: Analysephase, Requirements Engineering

- **Beispiele für bekannte allgemein einsetzbare Techniken zur Anforderungsermittlung/-analyse (v.a. im OO Bereich):**

(1) Fragenkataloge, Dokumentenmuster

(2) Hauptwortansatz (Noun Approach)
Identifikationshilfe für Klassen aus einer Problembeschreibung

(3) Use Cases / Use Case Szenarien (Anwendungsfälle)

(4) CRC Cards (Class-Responsiblity-Collaboration)

6.10 Fokus: Analysephase, Requirements Engineering

Hauptwortansatz (Noun Approach)

Die Stadtverwaltung einer Großstadt will die Müllabfuhr der Stadt rechnergestützt organisieren. Dazu werden Entsorgungsteams gebildet, die aus 3 bis 4 Mitarbeitern bestehen. Jedem Team ist ein Entsorgungsfahrzeug fest zugeordnet. Ein Team ist für die Müllabfuhr in einem oder mehreren Bezirken zuständig. Die genaue Anzahl der von einem Team zu versorgenden Bezirke ist von der Größe des Bezirks abhängig. Es gibt Bezirke der Größe 1, 2 oder 3. Die Summe dieser Größenfaktoren darf pro Team nicht größer als 6 werden, ein Team kann also z. B. 3 Bezirke der Größe 2 entsorgen.

6.10 Fokus: Analysephase, Requirements Engineering

UNIVERSITÄT OSNABRÜCK

Hauptwortansatz (Noun Approach)

Die Stadtverwaltung einer Großstadt will die Müllabfuhr der Stadt rechnergestützt organisieren. Dazu werden Entsorgungsteams gebildet, die aus 3 bis 4 Mitarbeitern bestehen. Jedem Team ist ein Entsorgungsfahrzeug fest zugeordnet. Ein Team ist für die Müllabfuhr in einem oder mehreren Bezirken zuständig. Die genaue Anzahl der von einem Team zu versorgenden Bezirke ist von der Größe des Bezirks abhängig. Es gibt Bezirke der Größe 1, 2 oder 3. Die Summe dieser Größenfaktoren darf pro Team nicht größer als 6 werden, ein Team kann also z. B. 3 Bezirke der Größe 2 entsorgen.

Markierung der Hauptwörter und Sammlung $\Rightarrow$ Klassenkandidaten:

Anzahl, Bezirke, Bezirken, Bezirks,
Entsorgungsfahrzeug, Entsorgungsteams,
Größe, Größenfaktoren, Großstadt,
Mitarbeitern, Müllabfuhr, Stadt,
Stadtverwaltung, Summe, Team

6.10 Fokus: Analysephase, Requirements Engineering

UNIVERSITÄT OSNABRÜCK

Hauptwortansatz (Noun Approach)

Die Stadtverwaltung einer Großstadt will die Müllabfuhr der Stadt rechnergestützt organisieren. Dazu werden Entsorgungsteams gebildet, die aus 3 bis 4 Mitarbeitern bestehen. Jedem Team ist ein Entsorgungsfahrzeug fest zugeordnet. Ein Team ist für die Müllabfuhr in einem oder mehreren Bezirken zuständig. Die genaue Anzahl der von einem Team zu versorgenden Bezirke ist von der Größe des Bezirks abhängig. Es gibt Bezirke der Größe 1, 2 oder 3. Die Summe dieser Größenfaktoren darf pro Team nicht größer als 6 werden, ein Team kann also z. B. 3 Bezirke der Größe 2 entsorgen.

Markierung der Hauptwörter und Sammlung $\Rightarrow$ Klassenkandidaten:

Anzahl, Bezirke, Bezirken, Bezirk,
Entsorgungsfahrzeug, Entsorgungsteams,
Größe, Größenfaktoren, Großstadt,
Mitarbeitern, Müllabfuhr, Stadt,
Stadtverwaltung, Summe, Team

Elimination redundanter Begriffe bzw. von Synonymen

Hauptwortansatz (Noun Approach)

- Physische Objekte sind gute Klassenkandidaten (z.B. „Mitarbeiter“)
- Eigennamen bezeichnen oft Objekte und identifizieren daher oft Klassen, wenn mehrere Objekte und eine eigenständige Behandlung gegeben sind
- Abstrakte Objekte (nicht physische) sind häufig mit physischen Objekten verbunden (Sicht auf physische Objekte, z.B. Größe, Team)
- Ein Klassenkandidat ohne Attribut ist meist ein Attribut einer anderen Klasse. Klassenkandidaten mit eigenen Attributen sind meist Klassen.
  - **Aufwendig (Vielzahl von Klassen ohne Beziehungen, Reduktion)**
  - **Hohes Fachwissen erforderlich (Fachtermini)**
  - **Einfach und universell**

6.10 Fokus: Analysephase, Requirements Engineering

- **Unterstützung der Tätigkeiten des Requirements Engineering im OO-Vorgehen v.a. durch Anwendungsfälle (Use Cases) und CRC-Cards sowie die schon bekannten Modelle (z.B. Klassendiagramm)**

- **Beispiel: Anwendungsfall Kreditvergabe**

  Use Case Diagramm +
  textuelle Use Case Diagrammbeschreibung +
  Szenariodiagramme

Name	Kredit vergeben
Akteure	Kunde, Schalterangestellte, Sachbearbeiterin, SCHUFA, Abteilungsleiterin
Beschreibung	Der Anwendungsfall beschreibt die Vergabe eines Privatkredites von der Antragstellung am Schalter bis zur Gewährung oder Ablehnung.
Startereignis	Der Kunde meldet sich am Bankschalter.
Ausnahmen & Fehler	Die Kreditsumme ist größer als 10.000 Euro.

6.10 Fokus: Analysephase, Requirements Engineering

Typischer Verlauf

- Der Kunde teilt der Schalterangestellten mit, dass er einen Kredit beantragen will.
- Kunde und Schalterangestellte füllen zusammen das Kreditantragsformular aus.
- Die Schalterangestellte gibt das Kreditantragsformular an die Sachbearbeiterin.
- Die Sachbearbeiterin gibt die Daten Kundenname, Kontonummer, Kreditsumme, Laufzeit, Zinssatz, Bürgschaften und monatliches Einkommen ins System ein.

6.10 Fokus: Analysephase, Requirements Engineering

- Das System fordert die SCHUFA-Auskunft an.
- Die SCHUFA liefert das Ergebnis der Überprüfung.
- Das System prüft die Kreditvergabe nach den hausinternen Richtlinien.
- Wenn die SCHUFA-Auskunft und die Prüfung nach den hausinternen Richtlinien positiv ausgehen, schickt das System dem Kunden eine Kreditgewährungszusage und schreibt die Kreditsumme auf dem Konto gut.
- Ansonsten schickt das System einen Kreditablehnungsbrief an den Kunden.

6.10 Fokus: Analysephase, Requirements Engineering

Alternativer Verlauf Kreditsumme über 10.000 Euro

- (Anfang wie oben)
- Wenn die SCHUFA-Auskunft und die Prüfung nach den hausinternen Richtlinien positiv ausgehen, legt das System den Vorgang der Abteilungsleiterin vor.
- Die Abteilungsleiterin prüft den Kreditantrag und gibt ihre Entscheidung ins System ein.
- Falls die Entscheidung positiv ist, schickt das System dem Kunden einen Kreditgewährungsbrief und schreibt die Kreditsumme auf dem Konto gut.
- Ansonsten schickt das System einen Kreditablehnungsbrief an den Kunden.

6.10 Fokus: Analysephase, Requirements Engineering

- **CRC-Karten (Class-Responsibilities-Collaborators)**
  Eine CRC-Karte beschreibt Verantwortlichkeiten von Rollen/Objekten und die Zusammenarbeit mit anderen Rollen/Objekten.

  - Verantwortlichkeiten:
    Berechnen, speichern, eingeben, ausgeben, kontrollieren
  - Zusammenarbeit:
    Teilverantwortlichkeiten regeln, delegieren

- **Ziele:** - Kommunikationsbasis
  - Verteilung von Zuständigkeiten
  - Identizierung der Zusammenarbeit zwischen Rollen/Klassen

- Karteikarten als Ausgangsmaterial zur statischen und dynamischen Anordnung von Rollen/Klassen/Objekten auf höherer Abstraktionsebene als Klassendiagramme

6.10 Fokus: Analysephase, Requirements Engineering

- **Beispiel:**

Kreditantrag	
Verantwortlichkeiten	Zusammenarbeit
speichere den Bearbeitungszustand	
sich selbst an Bankangestellten delegieren	Kreditbearbeitung
SCHUFA-Auskunft einholen	SCHUFA
Korrespondenz	Postausgang

Kreditbearbeitung	
Verantwortlichkeiten	Zusammenarbeit
nehme Antrag entgegen	Kreditantrag
prüfe Antrag auf Konsistenz	Richtlinien
leite Antrag ggf. weiter	Abteilungsleiterin

6.10 Fokus: Analysephase, Requirements Engineering

- **Vorgehen:**

  - An Hand der Anwendungsfälle werden die CRC-Karten erarbeitet.
  - **Identifikation der Objekte: Substantive in Anwendungsfällen**
  - **Zustände und Verantwortlichkeiten festlegen: Adjektive, Verben**
  - **Anlegen der Rolle-/Klassenkarten**
  - **Erfassung und Analyse der Zusammenarbeit mit anderen Objekten**

  - Nach einer ersten Beschreibung iteratives Vorgehen:
  - **Sind alle benötigten Verantwortlichkeiten verteilt?**
  - **Gibt es Überlappungen? → Auflösen, Generalisierung (Vererbung)**
  - **Ziel: hohe Kohäsion und lose Kopplung, Verantwortlichkeitsprinzip**

  - Aus den CRC-Karten erstes Klassendiagramm erzeugen.

  - Anwendungsfälle in Sequenzdiagramme überführen.

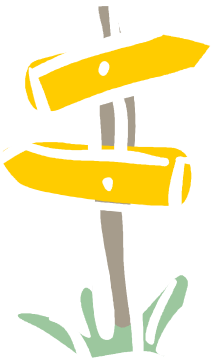
  - Weitere Diagramme entwerfen, z.B. Lebenszyklen von Objekten (Zustandsdiagramme).

Softwareprozesse, -vorgehensmodelle, -lebensläufe

Zusammenfassung und Ausblick

1 **Software-Krise und Software Engineering**
2 **Grundlagen des Software Engineering**
3 **Projektmanagement**
4 **Konfigurationsmanagement**
5 **Software-Modelle**
6 **Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle**
7 **Qualität**
8 **… Fortgeschrittene Techniken**

6.1 Softwareentwicklung in Phasen
6.2 Unsystematische “Modelle”
6.3 Lineare, sequentielle Modelle
6.4 Frühe Prototypen (Rapid Prototyping)
6.5 Evolutionäre, inkrementelle Modelle
6.6 Objektorientierte Modelle
6.7 Microsoft Vorgehen
6.8 Agile Modelle
6.9 Weitere Phasenmodelle
6.10 Fokus: Analysephase, Requirements Engineering



→ **Wege im Umgang mit der Software-Krise und Umsetzung der Grundlagen und Prinzipien:**
Einsatz von Vorgehensmodellen zur Anordnung der SW Entwicklungsphasen & Qualitätssicherung