

# 1 Agile Softwareentwicklung aus Auftraggebersicht \*

**Wichtig – Unwichtig – Egal?**

**Prof. Dr. Helmut Balzert**



»Didaktik ist unsere Stärke«

Folien zum Vortrag »Agile Softwareentwicklung aus Auftraggebersicht«

Treffpunkt@IT-Ruhr am 26.11.2009

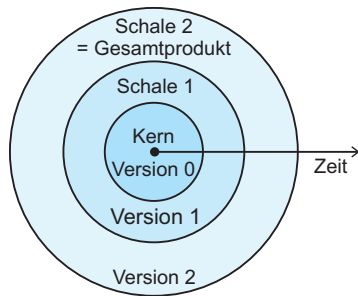
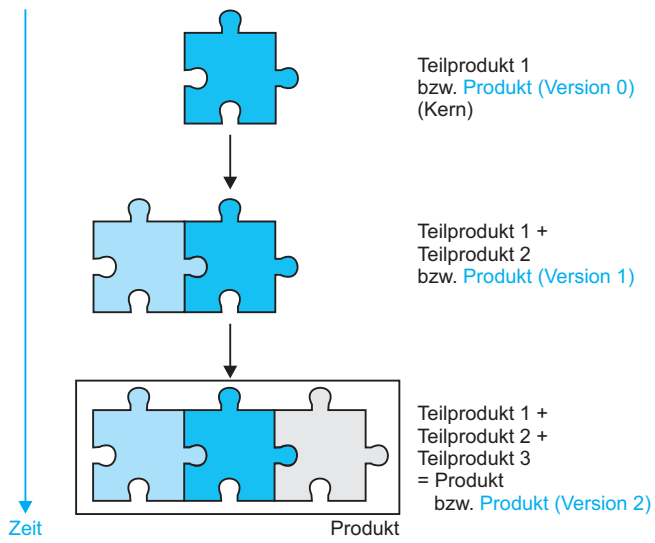
Weitere Informationen finden Sie unter [www.it-ruhr.de](http://www.it-ruhr.de).

© W3L GmbH 2009

## **2 Gliederung \***

- Inkrementell – Iterativ
- Agile Modelle
- Beispiele für agile Modelle
- Resümee

### 3 Inkrementell - Iterativ \*



## 4 Agile Modelle \*

### ■ Kennzeichen

- Kompromiss zwischen *keinem* Prozess und *zuviel* Prozess, so dass gerade soviel Prozess vorhanden ist, damit sich der Aufwand lohnt
- Möglichst wenig Dokumente, im Extremfall ist der Code das Dokument.
- Iterative Entwicklung mit häufig ausgelieferten lauffähigen Versionen des Zielsystems.

Lauffähige Versionen besitzen oft nur wenig Funktionalität, sollten aber bereits Vertrauen in das endgültige System geben.

Lauffähige Versionen sollen voll integriert und so sorgfältig getestet werden wie eine endgültige Auslieferung.

- Stabile Pläne sind Pläne für *kurze* Zeiträume und werden für jeweils eine einzelne Iteration gemacht.

### ■ Auswahl agiler Modelle

- XP: eXtreme Programming
- IXP: Industrial XP
- XP2: eXtreme Programming 2
- FDD (*Feature Driven Development*)
- SCRUM
- Crystal Family*

## 5 Manifest agiler Software-Entwicklung \*\*

- **Einzelpersonen und Interaktionen**  
wichtiger als Prozesse und Werkzeuge
- **Laufende Systeme**  
wichtiger als umfassende Dokumentation
- **Zusammenarbeit mit dem Kunden**  
wichtiger als Vertragsverhandlungen
- **Reaktion auf Änderungen**  
wichtiger als das Verfolgen eines Plans

### Prinzipien: Die agile Allianz

- Unsere höchste Priorität liegt darauf, den **Kunden** durch frühzeitige und kontinuierliche Auslieferung einsetzbarer Software **zufriedenzustellen**
- **Wir begrüßen sich änderende Anforderungen**, auch in einem späten Entwicklungsstadium
- Agile Prozesse werden geändert, damit der Kunde einen Wettbewerbsvorteil hat
- **Wir liefern häufig lauffähige Software** in einer Zeitspanne von mehreren Wochen und mehreren Monaten aus, wobei wir kürzere Zeiten bevorzugen
- Management und Entwickler müssen täglich im Projekt zusammenarbeiten
- Projekte basieren auf motivierten Mitarbeitern
- Die effizienteste und effektivste Methode um Informationen in einem Entwicklungsteam zu vermitteln ist von Angesicht zu Angesicht
- Lauffähige Software ist das primäre Fortschrittsmaß
- Agile Prozesse unterstützen eine nachhaltige Entwicklung. **Auftraggeber**, Entwickler und Benutzer sollen in der Lage sein, die **Entwicklung permanent mitzuvollziehen**
- Die ständige Beachtung technischer Perfektion und guter Entwürfe unterstützt die Flexibilität
- Einfachheit – die Kunst nichtnotwendige Arbeit nicht zu tun – ist essenziell
- Die besten Architekturen, Anforderungen und Entwürfe stammen von sich selbst organisierenden Teams
- In regelmäßigen Abständen reflektiert das Team darüber, wie es effektiver werden kann und paßt das eigene Verhalten dann entsprechend an

## 6 XP: eXtreme Programming \*

### 4 Werte

- Einfachheit
- Kommunikation
- Rückkopplung
- Mut

### 15 Prinzipien

- Unmittelbare Rückkopplung
- Einfachheit anstreben
- Inkrementelle Änderungen
- Neues wollen
- Qualitätsarbeit
- Lernen lehren
- Geringe Anfangsinvestition
- Auf Sieg spielen
- Gezielte Experimente
- Offene, ehrliche Kommunikation
- Team-Instinkte nutzen, nicht dagegen arbeiten
- Verantwortung übernehmen
- An lokale Rahmenbedingungen anpassen
- Mit leichtem Gepäck reisen
- Ehrliches Messen

### Praktiken

- 14 XP-Praktiken
- **Kunde vor Ort** (*on-site customer*):
  - Kunden und Anwender (kein Unterschied bei XP) stehen dem Entwicklungsteam als Ansprechpartner für fachliche Fragen in der Regel permanent zur Verfügung
  - Sie schreiben kein Pflichtenheft oder ähnliches.
  - Im Rahmen der QS übernehmen Akzeptanztests in übertragenem Sinne die Rolle der Anforderungsspezifikation.

### Contra 1

- Kunden (Auftraggeber) und Anwender (Benutzer) haben in der Regel unterschiedliche Ziele.
- Während für den Auftraggeber beispielsweise die Funktionalität im Vordergrund steht, ist für den Benutzer z. B. die Bedienungsfreundlichkeit das wichtigste Kriterium.

<b>Management-praktiken</b>	<b>Teampraktiken</b>	<b>Programmier-praktiken</b>
<b>Kunde vor Ort</b>	Metapher	Testen
<b>Planungsspiel</b>	Gemeinsame Verantwortlichkeit	Einfacher Entwurf
Besprechungen im Stehen	Fortlaufende Integration	Re-Strukturierung
<b>Kurze Releasezyklen</b>	Programmierstandards	Paarweises Programmieren
Retrospektiven	Nachhaltiges Tempo	

- Daher muss sichergestellt sein, dass beide Gruppen vertreten sind, was durch den XP-Ansatz *nicht* gewährleistet wird.

### **Contra 2**

- Nicht jeder Auftraggeber ist bereit oder in der Lage, permanent einen Ansprechpartner für das Entwicklungsteam bereitzustellen.
- Für kleinere Projekte ist das für den Auftraggeber auch unwirtschaftlich.

### **Contra 3**

- Der Kunde vor Ort kann auftretende Fragen auch *nicht* immer schnell und verlässlich klären und ist oft schlecht ausgelastet – oft nur zu einem Viertel seiner Zeit (vgl. /Padberg, Tichy 07, S. 166/).

#### ■ **Planungsspiel** (*planning game*):

- XP-Projekte sind iterativ und inkrementell
- Im sogenannten Planungsspiel wird der Umfang des jeweils nächsten Inkrements zwischen Kunden, Anwendern und Entwicklern festgelegt.
- Anwender und Kunden geben die Prioritäten der zu realisierenden Anforderungen an, die Entwickler schätzen die Aufwände.

#### ■ **Kurze Releasezyklen** (*short releases*):

- Neue und geänderte Teilprodukte werden den Anwendern in kurzen Abständen zur Verfügung gestellt.
- Die ersten Erfahrungen der Anwender werden bei der Weiterentwicklung berücksichtigt.

#### ■ **Einfacher Entwurf** (*simple design*):

- Das zu entwickelnde Produkt soll möglichst einfach gestaltet werden, da einfache Entwürfe schneller realisiert werden können und leichter zu verstehen sind .

### **Contra**

- Komplexe Probleme führen in der Regel auch zu komplexen Lösungen.

## ■ Paarweise Programmierung (*pair programming*)

### Contra 1

- Nicht jeder ist auf die Dauer ein Teamplayer und kann sich im Team konzentrieren.
- Alle, die als Einzelne top oder in ihrer Leistungsfähigkeit beschränkt sind, werden bei XP ausgeschlossen

### Contra 2

- Ist der Weg zur Problemlösung klar, dann arbeitet ein Einzelner alleine oft schneller.
- Eine paarweise Arbeit kostet dann Zeit und Geld.

### Contra 3

- Zurzeit ist noch nicht klar, wann zwei Entwickler ein gutes Paar bilden.
- Sie dürfen sich in der Qualifikation nicht zu stark unterscheiden, sonst dominiert der erfahrene Entwickler den unerfahrenen.
- Ist die Qualifikation zu ähnlich, dann entstehen keine neuen Ideen und beide Entwickler neigen zu denselben Fehlern (vgl. /Padberg, Tichy 07, S. 164 f./).

### Bewertung

- + Alle Abläufe werden auf die eigentliche Wertschöpfung bei der Softwareentwicklung ausgerichtet.
- + Alle Prinzipien und Praktiken helfen, sich flexibel auf Kundenwünsche einzustellen.
- XP funktioniert nur dann gut, wenn eine ganze Reihe von Randbedingungen, wie Teamgröße, Persönlichkeitsprofil und Qualifikation der Teammitglieder, zutreffen.
- Testfälle ersetzen die Anforderungsspezifikation und auch den Entwurf. Anforderungs- und Entwurfsprobleme werden iterativ direkt beim Programmieren gelöst.
- Es ist unklar, ob die häufigen Umstrukturierungen (*Factoring*) der Software, die sich durch die Anforderungsentwicklung parallel zur Programmierung ergeben, Zeit und Qualität positiv beeinflussen.
- XP verhindert, dass fachliche Problemlösungen, z. B. mit Hilfe der UML, auf einem höheren Abstraktionsniveau erfolgen – unabhängig von der konkreten technischen Umgebung und Realisierung.
- Ein Entwickler muss alles können: Analysieren, Entwerfen, Programmieren und Testen. Eine Spezialisierung – wie sie bei dem zunehmenden Wissenszuwachs auf allen diesen Gebieten immer notwendiger wird – wird dadurch verhindert.
- XP geht – zumindest implizit – von einem Teamgedanken und einem Menschenbild aus, die in etwa so aussehen:
- Kleines Team, Teammitglieder auf derselben »Wellenlänge« und dem gleichen intellektuellen Niveau, alle arbeiten in einem Raum, alle denken sehr programmiernah.

## 7 XP2: eXtreme Programming 2 \*\*\*

- Weiterentwicklung von XP – 2004

### Werte

- Einfachheit, Kommunikation, Rückkopplung, Mut + **Respekt**

### Primärpraktiken

- **Beieinander sitzen:**

Alle Projektbeteiligten sitzen räumlich nahe beieinander – idealerweise in einem Raum.

### Contra

- Programmieren erfordert eine hohe Konzentration.  
Umgebungsgeräusche sind kontraproduktiv.
- **Energiegeladene Arbeit** : Alle Teammitglieder sind engagiert bei der Arbeit.

### Contra

- Menschen sind wechselhaft, verändern sich von Tag zu Tag und von Ort zu Ort [Cockburn 99]
- **Paarweises Programmieren:** Unverändert gegenüber XP.
- **Geschichten:** Die Anforderungen werden als informelle Geschichten aufgeschrieben.
- **Wochenzyklus:**  
Jede Iteration dauert 1 Woche
- **Quartalszyklus:**  
Jedes Release dauert 3 Monate
- **Test vor Programmierung** : Vor der Programmierung werden die Tests erstellt.

### Contra 1

- Diese Praktik ist schwer zu vermitteln. Viele Programmierer haben Schwierigkeiten damit, die Reihenfolge von Codieren und Testen umzudrehen (vgl. [Padberg, Tichy 07, S. 165]).

### Contra 2

- Testen ist einer der am wenigsten effektiven Wege zur Fehlerbeseitigung. Inspektionen sind wesentlich effektiver [De Luca 07, S. 75].
- **Inkrementeller Entwurf:**  
Der Entwurf erfolgt schrittweise, getrieben von den aktuellen Anforderungen.  
Der Entwurf darf komplex sein.  
Ein großer Gesamtentwurf (*big design upfront*) ist aber *nicht* erlaubt

### Contra

- Beim Programmieren entsteht ein »inhärenter Entwurf«, der aus dem Code vielleicht extrahierbar ist.  
Es ist jedoch unklar, welche Qualität bzgl. Modularität und Wartbarkeit ein solcher Entwurf gegenüber einem vorab erstellten Entwurf hat.

## **Folge-Praktiken**

### **Echte Kundenbeteiligung**

#### **Inkrementelle Auslieferung:**

Erstellte Releases sollen beim Kunden auch eingesetzt werden, auch wenn nur Teile der Funktionalität verfügbar sind.

Sie sollten bereits parallel mit Altsystemen benutzt werden.

#### **Code und Tests:**

Nur der Quellcode und die Tests werden als Dokumentation erstellt und gepflegt.

## **Contra**

- Nach der Auslieferung muss eine Anwendung weiterentwickelt, gepflegt und angepasst werden.

Es ist fraglich, ob allein der Quellcode und die Tests geeignet sind, um diese Tätigkeiten wirtschaftlich durchzuführen.

Der Nutzen von UML-Entwurfsdiagrammen bei Wartungsaufgaben wurde durch eine empirische Studie nachgewiesen (vgl. [Padberg, Tichy 07, S. 166]).

#### **Tägliche Auslieferung:**

Täglich soll der aktuelle Systemstand an die Anwender ausgeliefert werden

#### **Vertrag mit aushandelbarem Umfang:**

Anstelle eines Festpreisvertrags wird ein Vertrag mit festem Budget und verhandelbarem Funktionsumfang abgeschlossen.

#### **Bezahlung pro Benutzung:**

Das System wird nicht pro Release bezahlt, sondern er werden Funktionen pro Benutzung abgerechnet

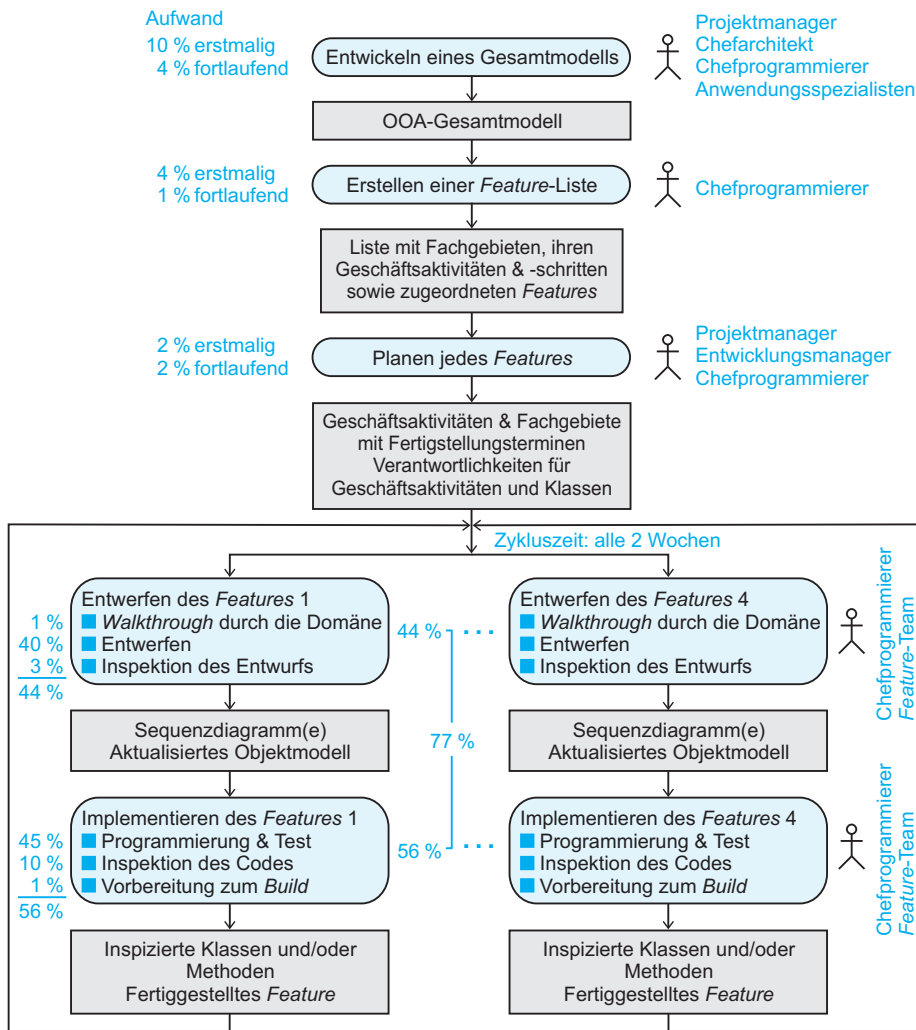
Kunde und Auftragnehmer ziehen so an einem Strang

Eine häufige Benutzung der Funktionen erzeugt für den Kunden einen hohen Geschäftswert und gleichzeitig Umsätze für den Auftragnehmer

## **Bewertung**

- + In einem dynamischen Umfeld mit nicht im Voraus bekannten Anforderungen, einem innovativen Produkt und einer kleinen bis mittleren Projektgröße reduzieren die Prinzipien und Praktiken von XP2 die Risiken bei einer Softwareentwicklung.
- + Durch die Konzentration auf die Kundenwünsche, auf den lauffähigen Code, auf die hinein entwickelte Qualität und kurze Auslieferungszyklen wird jeder unnötige Ballast vermieden.
- Viele Annahmen insbesondere bzgl. der Qualität, der Dokumentation und der Arbeitsweise sind empirisch nicht belegt.  
Es gibt gegenteilige Aussagen und Untersuchungen.
- In XP 2 werden für einzelne Entwicklungstätigkeiten absolute Zeitraster eingeführt (*time-boxed development*):  
Wochenzyklus für Iterationen, Quartalszyklus für Releases.  
Nachteilig daran ist, dass kritische Anforderungen oft nicht in solch festen Rastern umgesetzt werden können.  
Auch lässt sich eine zu umfangreiche Funktionalität nicht beliebig auf ein festes Zeitraster skalieren.

## 8 FDD - Feature Driven Development \*\*\*



Legende: OOA = objektorientierte Analyse

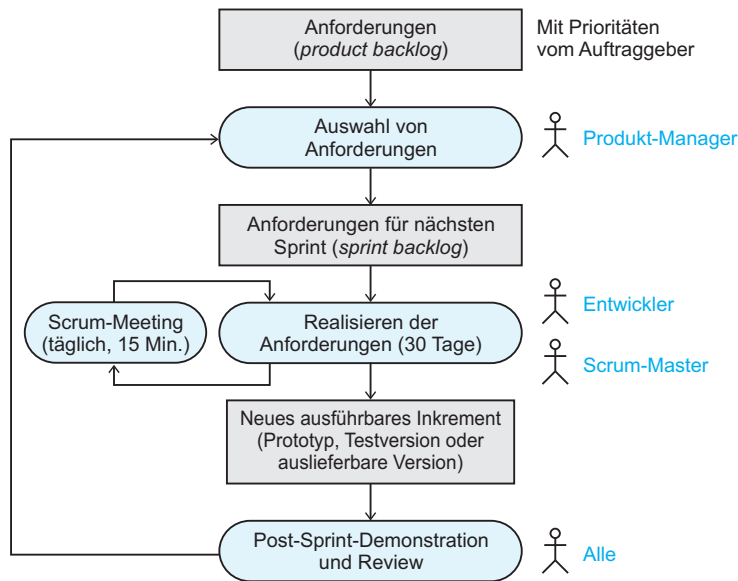
### ■ Vergleich mit XP

- Im Gegensatz zu XP gibt es für jede Klasse einen verantwortlichen Entwickler (*Class-Ownership*), während in XP jeder im Team jede Klasse ändern kann (*Collective-Ownership*).
- Statt paarweises Programmieren wie in XP verwendet FDD formale Inspektionen, die als wirksamer angesehen werden.
- Während es in XP *keine* Rollen gibt, gibt es in FDD einen Chefarchitekten und Chefprogrammierer. Sie verfügen über mehr Erfahrung und sind gleichzeitig Mentoren für die Entwickler.
- In XP legt der Auftraggeber die Prioritäten fest, in FDD legt der Auftraggeber die Prioritäten bei den Geschäftsaktivitäten fest, während die *Feature*-Prioritäten vom Chefprogrammierer nach technischen Gesichtspunkten bestimmt werden.
- Im Gegensatz zu XP wird in FDD zunächst ein grobes Gesamtmodell erstellt, um Überarbeitungsarbeiten durch spätere Erkenntnisse zu vermeiden (*»doing as much right the first time«*).
- Nur begrenztes Re-Strukturieren im Gegensatz zu XP.

## **Bewertung**

- + Gut geeignet für die objektorientierte Neuentwicklung von Individual- und Standardsoftware.
- + I. Allg Parallelarbeit mehrerer Teams in den Prozessen 4 und 5.
- Sequenzielle Abfolge der Prozesse; ein freier Wechsel zwischen den Prozessen ist nicht vorgesehen).

## 9 Scrum \*



Legende:



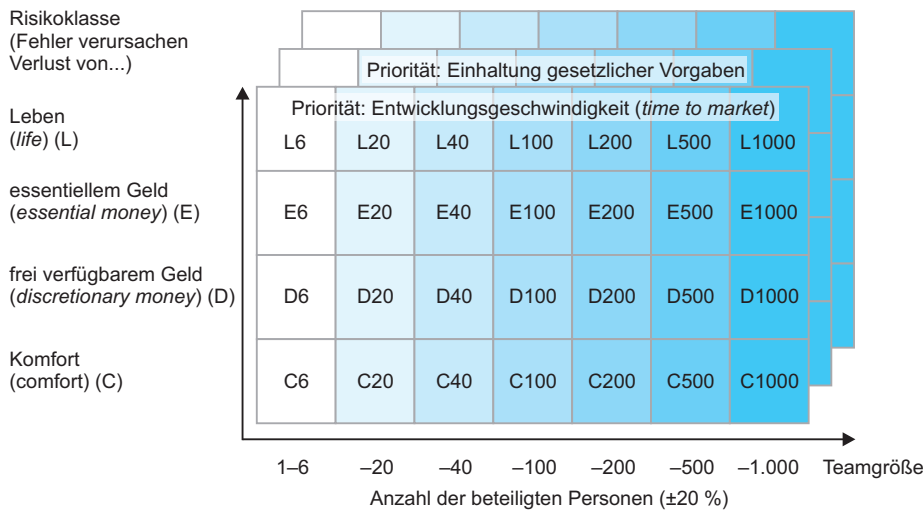
### Vergleich mit XP

- Scrum fokussiert auf das Projektmanagement, während XP sich auf die Programmierung konzentriert.
- Scrum setzt auf eine stärkere Eigenverantwortlichkeit der Entwickler verglichen mit XP.

### Bewertung

- + Gut geeignet, wenn die Anforderungen *nicht* im Voraus festgelegt werden können und »chaotische« Rahmenbedingungen antizipiert werden müssen.
- + Leicht erlernbar und schnell einsetzbar; auf ein paar Seiten beschreibbar.
- Erfolg hängt stark von den Teammitgliedern und ihrer Persönlichkeit ab.
- Keine Hilfestellungen für die Entwicklung von Software.

## 10 Die Crystal-Familie \*\*



### Vergleich mit XP

- Jeder Teil von XP kann durch *Clear* ersetzt werden, da XP alle *Clear*-Standards erfüllt, außer der Dokumentation.
- XP führt zu einer höheren Produktivität durch größere Disziplin, aber es ist für Teams schwerer XP zu befolgen /Highsmith 02, S. 267/.
- Crystal Clear* erlaubt eine größere Individualität im Team – u. U. auf Kosten der Produktivität (a. a. O.).
- Ein Team kann mit *Crystal Clear* starten und sich hin zu XP bewegen. Scheitert XP kann man zurück zu *Crystal Clear* (a. a. O.).

### Bewertung

- + *Crystal Clear* ist »leicht« (wenige methodische Elemente), tolerant (viele Variationen sind akzeptabel) und legt nur wenige Formalitäten fest (Dokumentation ist informal).
- + Das Baukastensystem hilft, sich auf projektgeeignete Prozessmodelle zu fokussieren. Es wird vermieden, dass ein 500-Mitarbeiter-Projekt im militärischen Bereich mit einem 6-Mitarbeiter-Web-Projekt verglichen wird.
- + Umkehrung des Prinzips des »Zuschneidens« (*Tailoring*). Statt ein umfassendes Modell »abzumagern«, wird aus einem Baukasten bei Bedarf – und nur bei Bedarf – ein Baustein hinzugenommen.
- Der Prozess-Bausteinkasten ist bisher nur rudimentär gefüllt.
- Extrem mitarbeiterorientiert, kann zu Problemen führen, wenn die Mitarbeiterprofile nicht so sind, wie *Crystal* es erwartet (kommunikativ, teamfähig).

## 11 Resümee \*

■ Agile Softwareentwicklung aus Auftraggebersicht:

Wichtig?

Unwichtig?

Egal?

## 12 Arabisches Sprichwort \*

Die Menschen lassen sich in drei Klassen einteilen:  
Diejenigen, die **unbeweglich** sind;  
diejenigen, die **beweglich** sind,  
und diejenigen, die sich **bewegen**.

## Literatur

/Agile Alliance 01/

*Manifesto for Agile Software Development*, 2001, <http://www.agilealliance.org/>.

Manifest für eine agile Softwareentwicklung.

/Ambler 06/

Ambler Scott W.; *Survey Says: Agile Works in Practice*, in: Dr. Dobb's Journal, September 2006, S. 62–64.

/BeAn04/

Beck, Kent; Andres, Dirk; *Extreme Programming Explained: Embrace Change*, 2. Auflage, Addison-Wesley, 2004.

/Beck 00/

Beck, Kent; *eXtreme Programming Explained*, Reading, Addison-Wesley, 2000.

/Coad, Lefebvre, De Luca 99/

Coad, Peter; Lefebvre, E.; De Luca, Jeff; *Java Modeling in Color with UML*, Prentice Hall, 1999.

/Cockburn 02/

Cockburn, Alistair; *Agile Software Development*, Boston, Addison-Wesley, 2002.

/Cockburn 97/

Cockburn, Alistair; *Surviving Object-Oriented Projects: A Managers Guide*, Addison Wesley Longman, 1997.

/Cockburn 99/

Cockburn, Alistair; *Characterizing People as Non-Linear, First-Order Components in Software Development*, 1999, <http://members.aol.com/humansandt/papers/nonlinear/nonlinear.htm>.

/Coldewey 01/

Coldewey, Jens; *Management am Rande des Chaos: Die Führung agiler Projekte*, in: OBJEKTSpektrum, 5/2001, 2001, S. 77–80.

/De Luca 07/

De Luca, Jeff; *Wir brauchen eine Ziellinie*, in: OBJEKTSpektrum, 5/2007, 2007, S. 73–76.

/Fowler, Highsmith 01/

Fowler, Martin; Highsmith, Jim; *The Agile Manifesto*, 2001, <http://www.sdmagazine.com/documents>.

Ausführliche Begründung des Manifests für eine agile Software-Entwicklung.

/Highsmith 02/

Highsmith, Jim; *Agile Software Development Ecosystems*, Boston, Pearson Education, Inc., 2002.

/Padberg, Tichy 07/

Padberg, Frank; Tichy, Walter; *Schlanke Produktionsweisen in der modernen Softwareentwicklung*, in: Wirtschaftsinformatik, 3, 2007, 2007, S. 162–170.

/Rising, Janoff 00/

Rising, L.; Janoff, N. S.; *The Scrum Software Development Process for Small Teams*, in: IEEE Software, July/August 2000, S. 26–32.

/Roock 07/

Roock, Stefan; *Feature Driven Development*, in: OBJEKTSpektrum, 5/2007, 2007, S. 65–66.

/Schwaber 04/

Schwaber, Ken; *Agile Project Management with Scrum*, Microsoft Press, 2004.  
Das aktuelle Standardwerk über Scrum.

/Wolf, Rock, Lippert 05/

Wolf, Henning; Rook, Stefan; Lippert, Martin; *eXtreme Programming*, 2. Auflage, Heidelberg, dpunkt-Verlag, 2005.