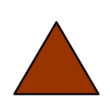
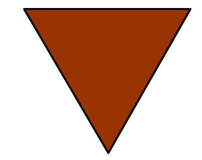
Einführung in agile Entwicklung

München, 24. Januar 2002

Jens Coldewey
Coldewey Consulting
Curd-Jürgens-Str. 4
D-81739 München
jens.coldewey@coldewey.com





Highsmith: We must challenge our most fundamental assumptions about software development

- Untersuchung von 63 erfolgreichen Projekten der 70er (z.B. IBM, GTE, TRW usw.)
- Die Kosten für Fehler steigen exponentiell
- Quintessenz: Fast jeder Aufwand ist gerecht-fertigt, um Fehler zu vermeiden

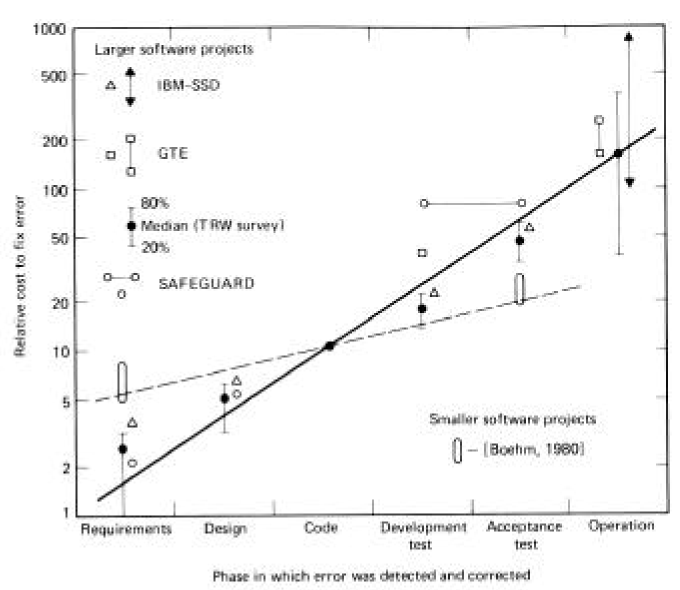
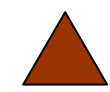


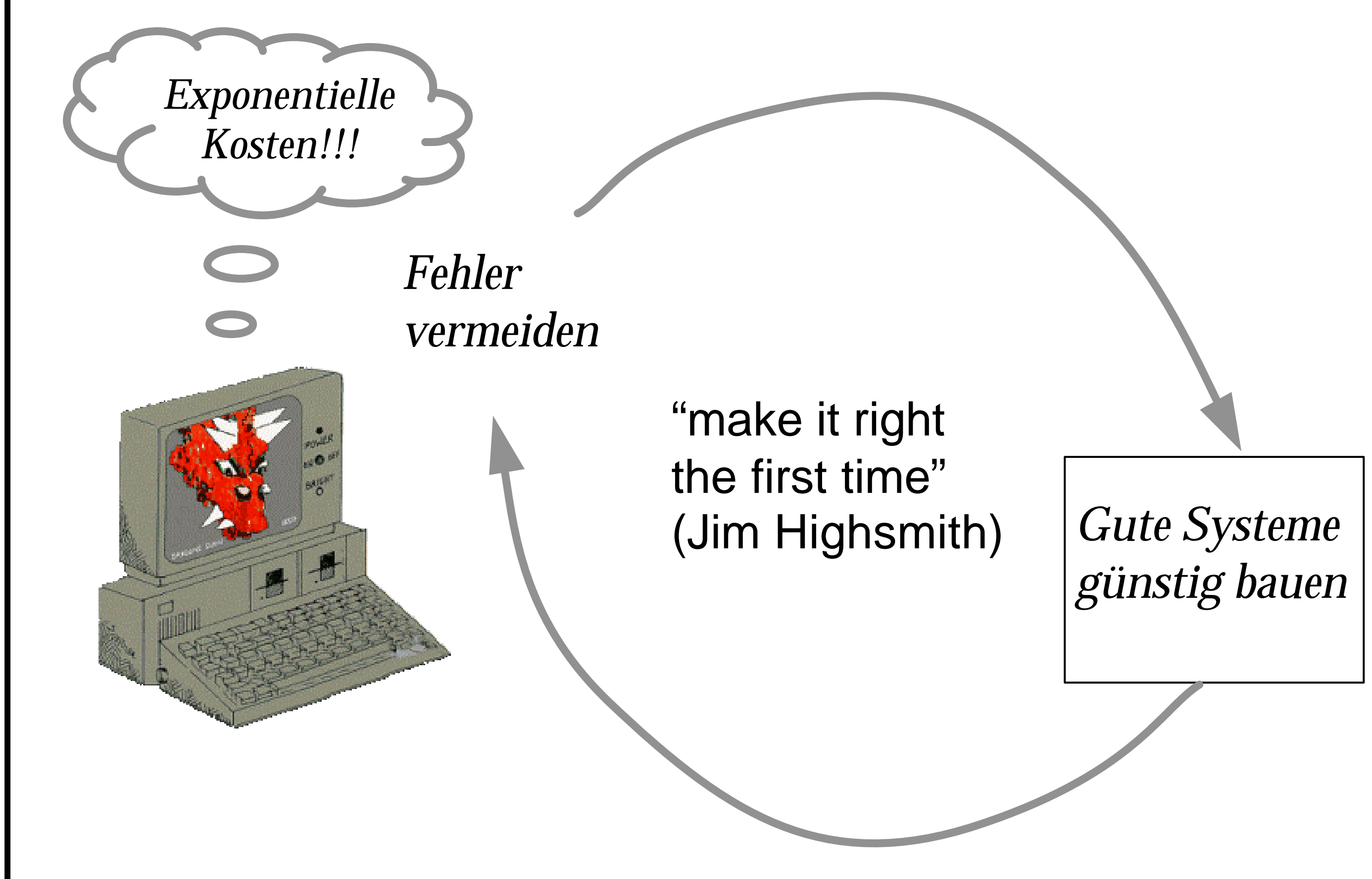
FIGURE 4-2 Increase in cost-to-fix or change software throughout life-cycle

Aus: Barry W. Boehm: *Software Engineering Economics* [Boe81], Seite 40

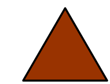
 ©Copyright: Coldewey Consulting 2001; All rights reserved

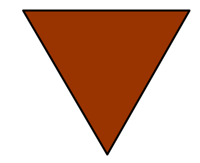


Fehler sind teuer - Ein Gegenmittel scheint, Fehler zu vermeiden

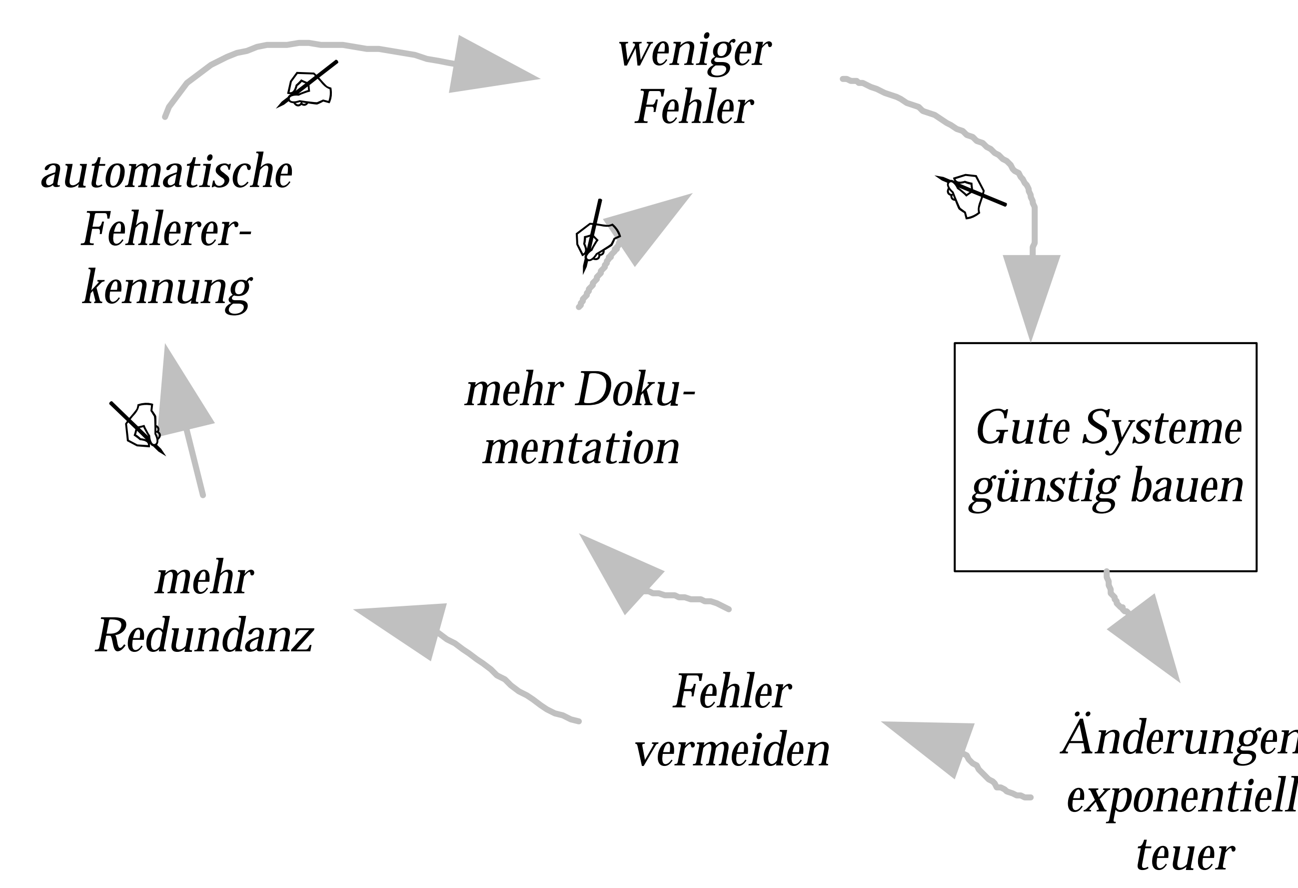


 ©Copyright: Coldewey Consulting 2001; All rights reserved





Traditionelle Prozesse erreichen dies Ziel mit Dokumentation und Redundanz



©Copyright: Coldewey Consulting 2001; All rights reserved

Warum sind Fehler so teuer?

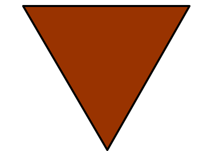
„If the ... error is not corrected until the maintenance phase, the correction involves a much larger inventory of specifications, code, user and maintenance manuals, and training material“

Aus: Barry W. Boehm: *Software Engineering Economics* [Boe81], Seite 39f

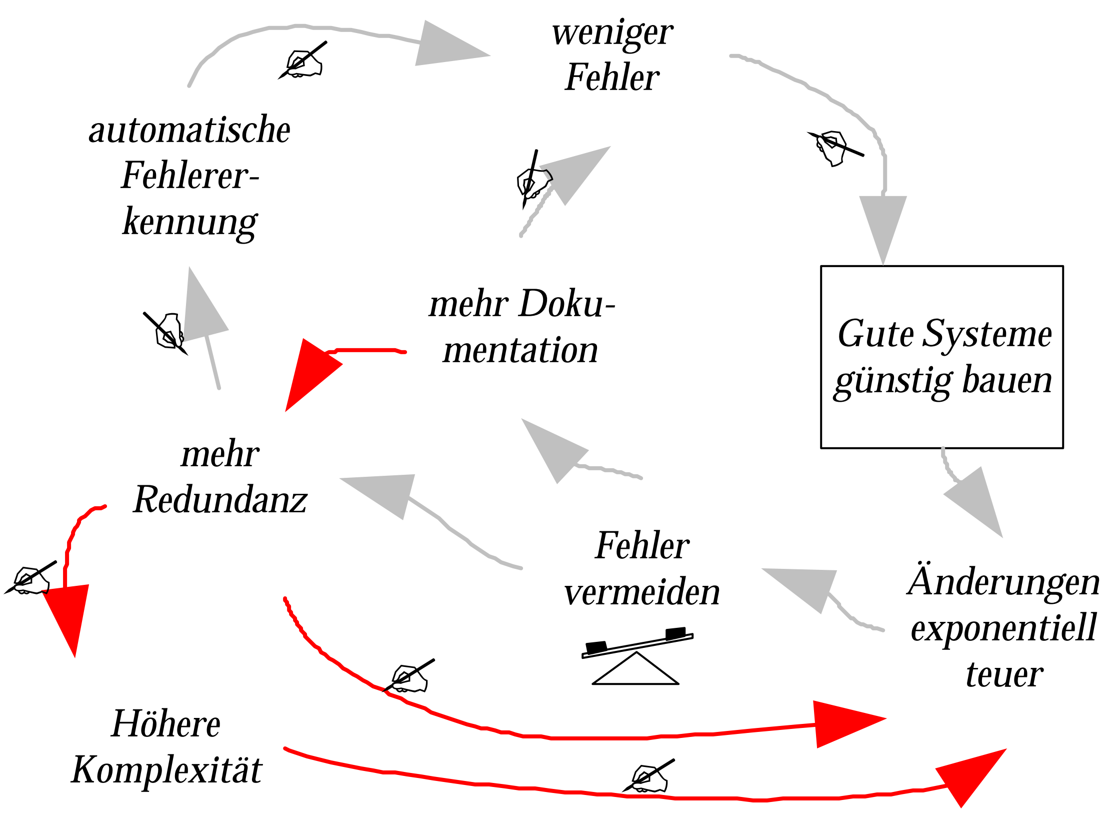
Systemanforderungen
Systemarchitektur
System-Integrationsplan
DV-Anforderungen
DV-Architektur
DV-Integrationsplan
SW-Anforderungen
SW-Architektur
Schnittstellenentwurf
SWKE-Integrationsplan
Datenkatalog
SW-Entwurf
Implementierungsdok. (Modul)
Implementierungsdok. (Komponente)
Implementierungsdok. (SWKE)
Benutzerhandbücher
Wartungshandbücher
Schulungsmaterial

R
e
d
u
n
d
a
n
z

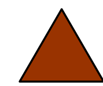
 ©Copyright: Coldewey Consulting 2001; All rights reserved



Offensichtlich muss die Systemsicht noch ergänzt werden



©Copyright: Coldewey Consulting 2001; All rights reserved



Was sind eigentlich Fehler?

Fokus traditioneller Prozesse

Fokus der Anwender

Spezifikation

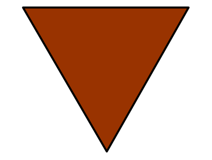
Produktion

Verifikation

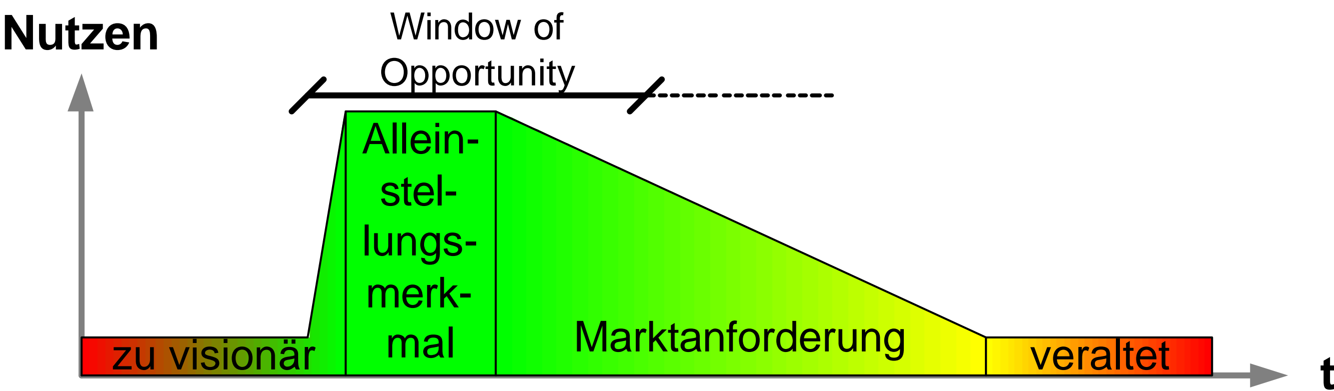
Validation

Fehler

 ©Copyright: Coldewey Consulting 2001; All rights reserved

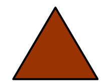


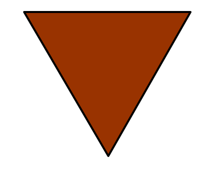
Betriebswirtschaftlich ist ein System nur in einer bestimmten Zeitspanne korrekt



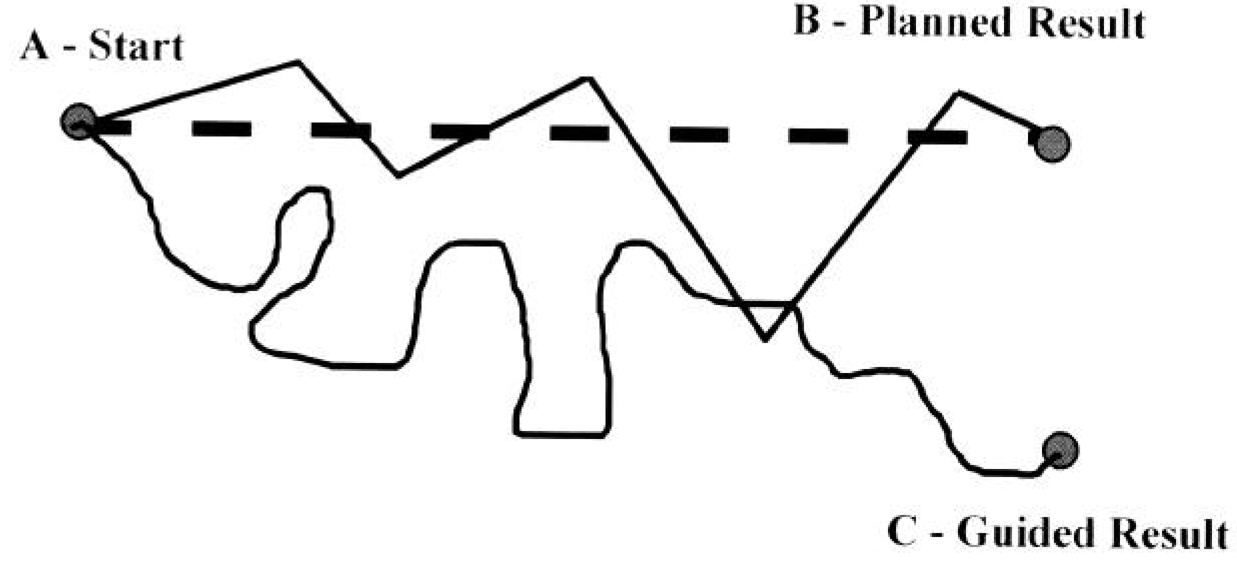
- Der Markt fordert heute schnelle Reaktionszeiten
- Häufig ändern sich die Marktanforderungen bereits deutlich während das System entwickelt wird
- Innovative Systeme sind häufig zu Projektbeginn noch unscharf
- Es ist kaum noch möglich, ein komplexes System im voraus vollständig und korrekt zu beschreiben

 ©Copyright: Coldewey Consulting 2001; All rights reserved





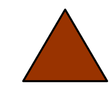
Ein vollständig spezifiziertes System ist nicht unbedingt das richtige System

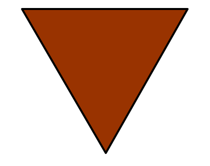


Aus: James A. Highsmith III: *Adaptive Software Development - A Collaborative Approach to Managing Complex Systems* [Hig00], Seite 43

„...following a plan produces the product you intended, just not the product you need“ - Jim Highsmith [Hig00]

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Projekte haben sich in den letzten 25 Jahren grundlegend geändert

Typische Projekte der Siebziger

- Abbildung stabiler Geschäftsprozesse
- Kein wesentlicher Innvoationsmotor
- Wenig technische Alternativen

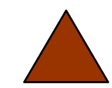
? „Moving Target“ war Zeichen schlechten Managements

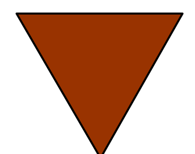
Typische Projekte heute

- Umstrukturierung der Geschäftsprozesse
- Wesentlicher Innovationsmotor
- Vielzahl leistungs-fähiger Entwicklungs-werkzeuge

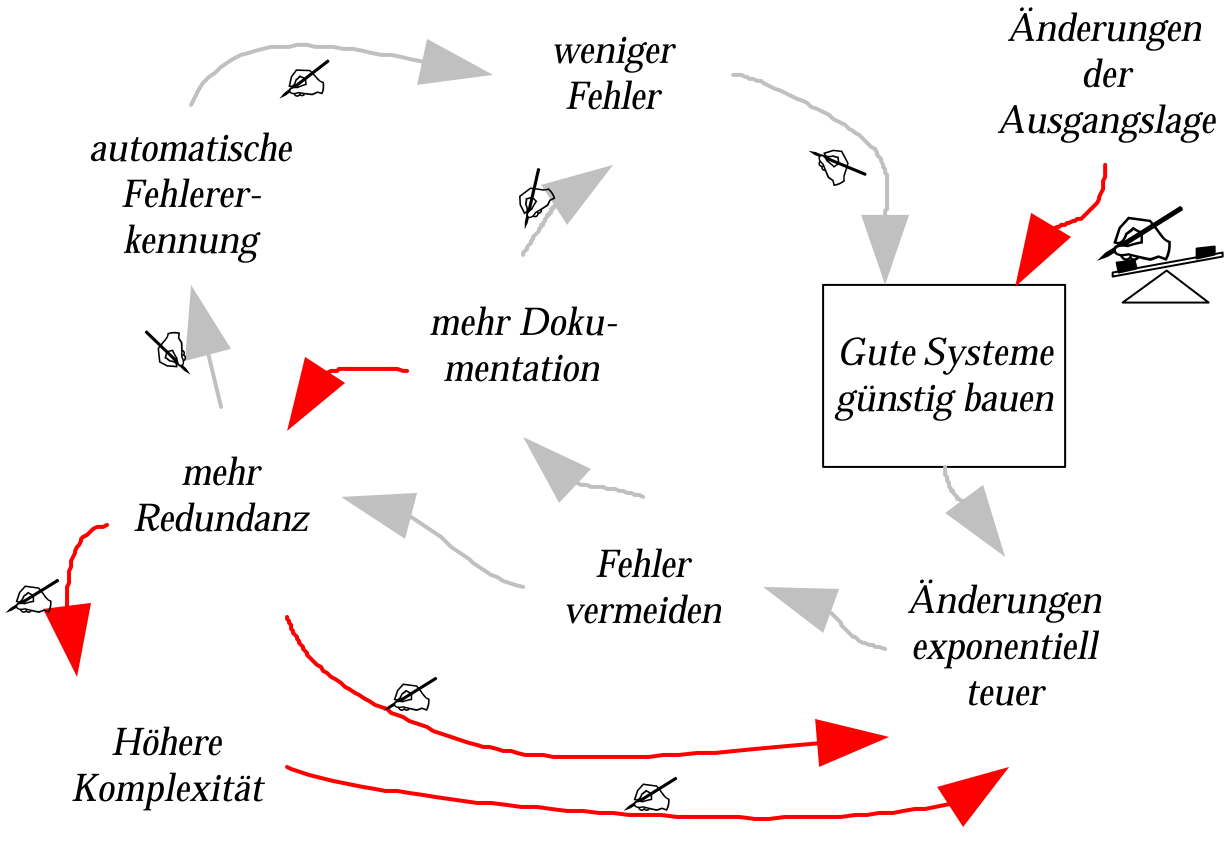
? „Moving Target“ ist die Regel

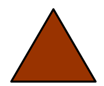
 ©Copyright: Coldewey Consulting 2001; All rights reserved



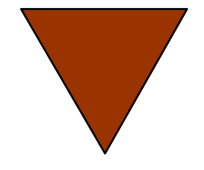


Die Systemsicht zeigt das Problem





 ©Copyright: Coldewey Consulting 2001; All rights reserved



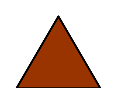
Das Problem hat sich geändert. Können wir uns die alten Lösungen noch leisten?

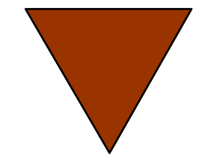
- Exponentielle Änderungskosten machen Wartung teuer
- Schwere Prozesse sind nicht nur (manchmal) Lösung, sondern auch Teil des Problems
- Lange Projektlaufzeiten können zum falschen System führen

? Was wäre, wenn Software änderbar wäre?

? Wie sieht die Entwicklung aus, wenn sie auf Änderbarkeit ausgerichtet ist?

 ©Copyright: Coldewey Consulting 2001; All rights reserved



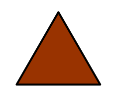


Teil II:
Die Grundprinzipien agiler Entwicklung

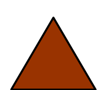
Software development is a cooperative game of invention and communication. He primary goal is to deliver working, useful software. The secondary goal […] is to set up for the next game

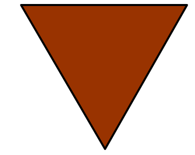
Alistair Cockburn [Coc02]

 ©Copyright: Coldewey Consulting 2001; All rights reserved



 ©Copyright: Coldewey Consulting 2001; All rights reserved



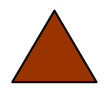


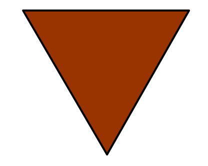
Kent Beck: „All methodolgy is based on fear“

Ängste agiler Entwicklung:

- Angst vor schlechter Zusammenarbeit
- Angst, zu spät oder gar keinen Code auszuliefern
- Angst, das Falsche zu entwickeln
- Angst, auf Änderungen nicht reagieren zu können

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Einzelpersonen und Interaktionen sind wichtiger als Prozesse und Werkzeuge

Prozesse sind wichtig

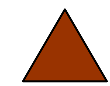
- Zur effizienten Abwicklung
- Zur Vergleichbarkeit
- Zur Vorhersehbarkeit
- Für Zertifizierungen

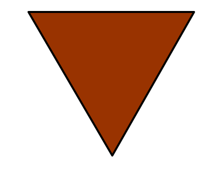
Aber Einzelpersonen und Interaktion sind wichtiger

- Miteinander reden ist die effizienteste Kommunikation
- Gute Arbeit erfordert gute Arbeiter
- Kreativität erfordert Motivation

Prozesse ersetzen kein Können und keine Ausbildung

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Laufende Systeme sind wichtiger als umfangreiche Dokumente

Dokumente sind wichtig:

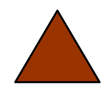
- Für nachfolgende Teams
- Um ein Thema auf zu arbeiten
- Um gemeinsames Verständnis zu erreichen
- Für rechtliche Vorgaben

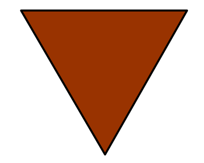
Aber laufende Software ist wichtiger

- Die Software ist die Realität, alles andere ist Modell
- Anwender verstehen Software, nicht UML
- Die Software muss das Geld verdienen

Dokumentation bedeutet nicht Verständnis

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Zusammenarbeit mit Kunden ist wichtiger als Vertragsverhandlungen

Verträge sind wichtig

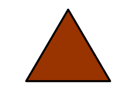
- Damit jeder weiß, was vereinbart ist
- Um bösen Überraschungen vorzubeugen
- Um beim Scheitern Verantwortung zu verteilen

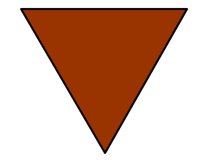
Aber Zusammenarbeit mit dem Kunden ist wichtiger

- Der Kunde weiß am besten was er braucht
- Viele Ideen entstehen erst im Projektverlauf
- Vor Gericht war noch kein Projekt erfolgreich

Es hat noch nie jemand einen Streit mit einem Kunden gewonnen

Folie 18; ©Copyright: Coldewey Consulting 2001; All rights reserved





Flexibilität ist wichtiger als Abarbeitung von Plänen

Planung ist wichtig

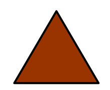
- Um Kosten abzuschätzen
- Zur Koordination
- Um Auslastung sicher zu stellen
- Um Fortschritt zu beurteilen

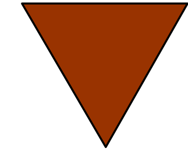
Aber Flexibilität ist wichtiger

- Anforderungen und Prioritäten ändern sich
- Pläne sind meist schlechte Modelle
- Wartung ist Änderung
- Software ist änderbar! (wenn man sie richtig baut)

Planung ist nicht Realität, Abweichung ist die Norm

 ©Copyright: Coldewey Consulting 2001; All rights reserved



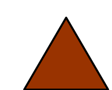


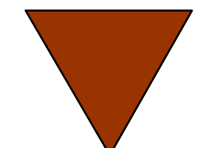
Teil III: Die Bausteine agiler Entwicklung

Our organizations work the way they work, ultimately, because of how we think and how we interact. Only by changing how we think we can change deeply embedded policies and practices

Peter Senge [Sen90]

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Agile Entwicklung baut auf fünf Bausteinen auf

Zusammen-
arbeit

Stilles Wissen

Änderbarkeit

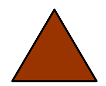
Simple rules lead to intelligent behavior, Complex rules lead to stupid behavior

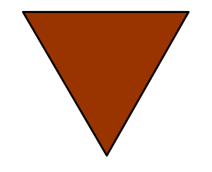
Jim Highsmith

Minimalismus

Kundenfokus

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Zusammenarbeit fördern: Abschied vom Kommandomanagement

Kommando-Management

- Ausgabe klarer Anweisungen an Einzelne
- Vorgabe des Arbeitsverfahrens
- Kontrolle der Ergebnisse
- Abweichungen vom Plan sind Zeichen schlechter Arbeit

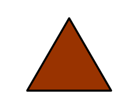
Follower

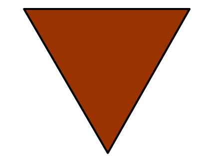
Agiles Management

- Gemeinsam getragene Vision, gemeinsame Planung der Arbeit
- Räumliche Nähe, direkte Kommunikation
- Konzentration auf Ergebnisse statt auf Verfahren
- Abweichungen vom Plan sind Zeichen weiterer Erkenntnis

Profis

 ©Copyright: Coldewey Consulting 2001; All rights reserved

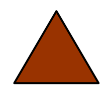


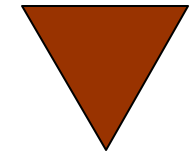


Um Gelerntes noch im Projekt nutzen zu können, wird inkrementell gearbeitet

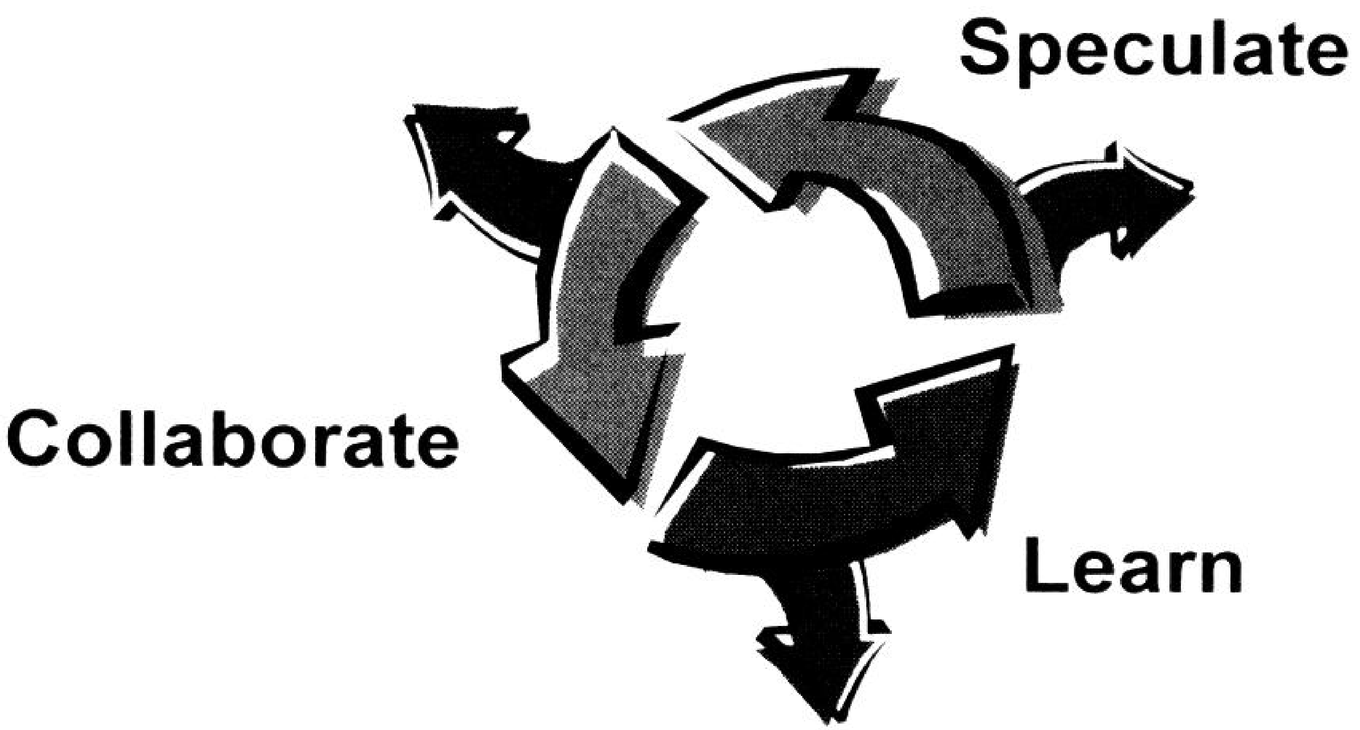
- Ausgeliefert wird in Inkrementen zwischen sechs und zwölf Wochen
- Jedes Inkrement muss zusätzlichen fachlichen Nutzen bringen
- Der Inhalt jedes Inkrements wird an seinem Anfang priorisiert und „festgelegt“
- Das Team beherrscht bald alle Phasen der Entwicklung sicher
- Technische Risiken werden frühzeitig ausgeschaltet
- Der Kunde bekommt schnell Vertrauen
- Neue Erkenntnisse können einfließen

 ©Copyright: Coldewey Consulting 2001; All rights reserved



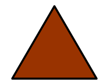


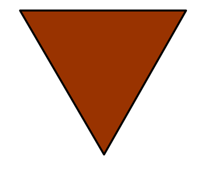
Aus dem überwachten Projekt wird ein „lernendes Team“



Aus: James A. Highsmith III: *Adaptive Software Development - A Collaborative Approach to Managing Complex Systems* [Hig00], Seite 41

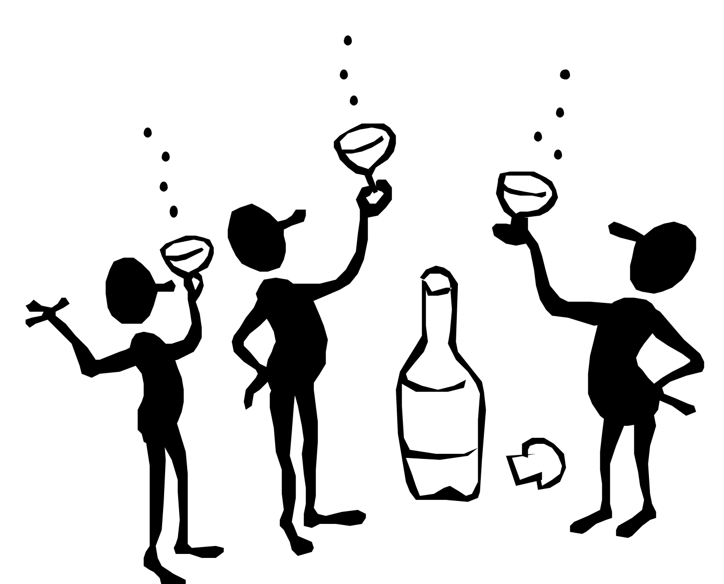
 ©Copyright: Coldewey Consulting 2001; All rights reserved



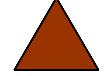


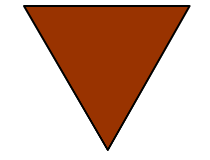
Stilles Wissen: Wie schmeckt ein gesundes Projekt?

- Schreiben bedeutet nicht gelesen werden, lesen bedeutet nicht verstehen
- Nur ein Bruchteil des Wissens ist dokumentierbar
- Profis urteilen meist „aus dem Bauch heraus“
- Dagegen zu arbeiten ist



Folie 25; ©Copyright: Coldewey Consulting 2001; All rights reserved

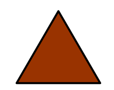


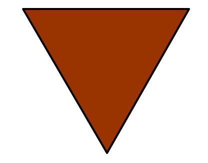


Stilles Wissen managen: Wissen verteilen statt Dokumente

- Der breiteste Kommunikationskanal ist die gemeinsame Arbeit
- Wer gemeinsam entwickelt muss nicht Dokumente lesen und (miss-)verstehen
- Wer weniger Dokumente schreiben muss hat mehr Zeit gute Software zu schreiben
- Wer zusammen entwickelt muss zusammen sitzen
- Wer Software schreibt, muss Software schreiben gelernt haben.
- Wer kommuniziert, muss kommunizieren gelernt haben
- Wer Expertin ist, muss Expertin geworden sein

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Wie Wissen transferiert wird, muss im Einzelfall entschieden werden

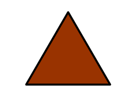
Schriftliche Dokumentation

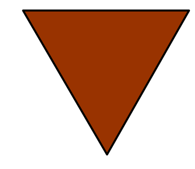
- \+ Reproduzierbar
- \+ Kontrollierbar
- \+ Hohe Streuung
- ✍ Vieles ist nicht dokumentierbar
- ✍ Keine Rückkopplung
- ✍ Schmalbandig
- ✍ Aufwändig
- ✍ Redundanz
- ✍ Scheinqualität (in? mm)

Kollaboration

- \+ Breitbandig
- \+ Schnell
- \+ Direkte Rückkopplung
- \+ Keine Redundanz
- ✍ Niedrige Streuung (n^2)
- ✍ Räumliche Nähe nötig
- ✍ Mangelnde Schulung
- ✍ Schwer Reproduzierbar
- ✍ Schwer Kontrollierbar

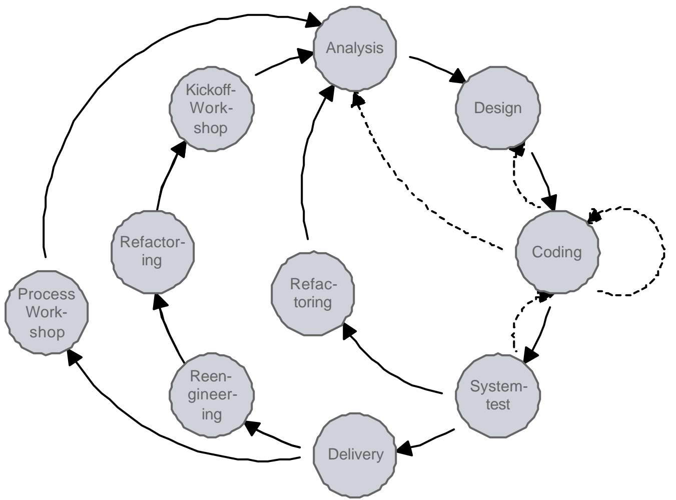
Folie 27; ©Copyright: Coldewey Consulting 2001; All rights reserved



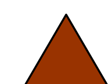


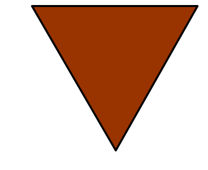
Das Team entscheidet, welche weiteren Dokumente notwendig sind

- Dokumentation ist im Code
- In regelmäßigen Workshops wird das Vorgehen analysiert und verbessert
- Dokumentiert wird, wenn das Team einen Vorteil darin sieht



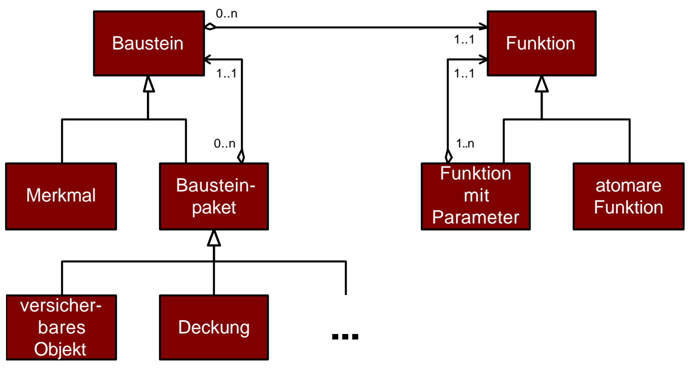
 ©Copyright: Coldewey Consulting 2001; All rights reserved



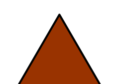


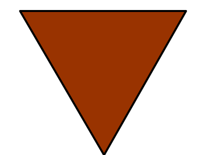
Eine gemeinsame Vision dient der fachlichen und technischen Konvergenz

- Eine Metapher skizziert das System fachlich
- Das Design dreht sich um wenige, zentrale Muster
- Technische Lösungen werden zugekauft
- Die Beschreibung erfolgt (wenn überhaupt) auf wenigen Seiten



 ©Copyright: Coldewey Consulting 2001; All rights reserved





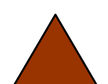
Wie macht man änderbare Systeme?

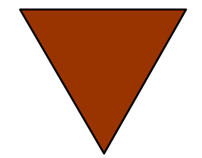
Redundanz vermeiden

Exponentielle Kosten

Gute Systeme günstig bauen

Folie 30; ©Copyright: Coldewey Consulting 2001; All rights reserved





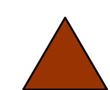
Redundanz vermeiden, bedeutet Konzentration auf das Wesentliche

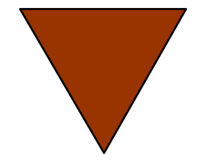
Systemarchitektur
System-Integrationsplan
DV-Anforderungen
DV-Architektur
DV-Integrationsplan
SW-Anforderungen
SW-Architektur
Schnittstellenentwurf
SWKE-Integrationsplan
Datenkatalog
SW-Entwurf
Implementierungsdok. (Modul)
Implementierungsdok. (Komponente)
Implementierungsdok. (SWKE)
Benutzerhandbücher
Wartungshandbücher

R e d u n d a n z

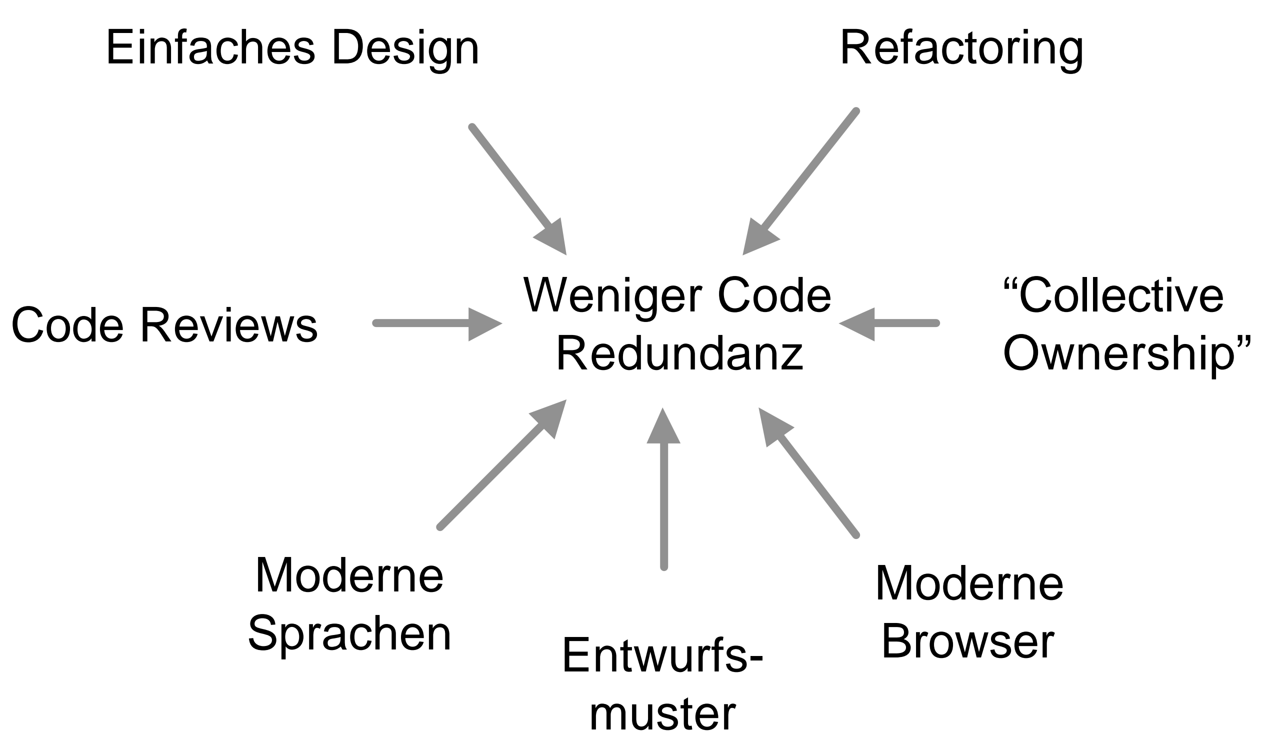
primär	sekundär
Code	
Anwenderdoku	
	Tests
	Verständnis der Domäne
	Verständnis des Systems

Folie 31; ©Copyright: Coldewey Consulting 2001; All rights reserved



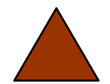


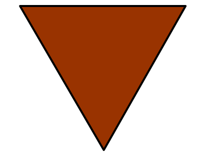
Code-Redundanz wird mit Werkzeugen entschärft und mit Design vermieden



“Do the most simple thing that might possibly work”
Kent Beck

©Copyright: Coldewey Consulting 2001; All rights reserved



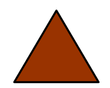


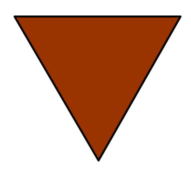
Regressionsfähige Tests stellen Änderbarkeit sicher

- Parallel zum Code werden automatisch ablaufende Testsuiten gebaut
- Fehler sind primär Lücken in Tests, die behoben werden müssen
- Eine Aufgabe ist erst dann abgeschlossen, wenn alle Tests durchlaufen

- Tests sind „nur einen Mausklick“ entfernt
- Relevante Tests laufen innerhalb von Sekunden durch
- Beim Refaktorieren bleiben die Testfälle unangetastet
- Testabdeckung bald höher als erwartet

 ©Copyright: Coldewey Consulting 2001; All rights reserved

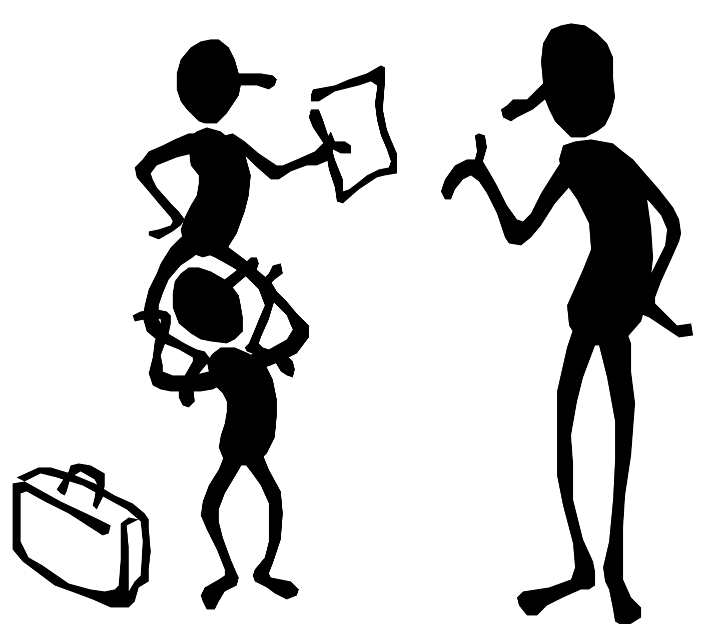




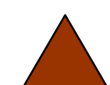
Kundenfokus: Alle wichtigen Entscheidungen werden gemeinsam getroffen

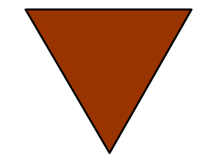
Regelmäßige Treffen mit (Geld- und) Auftraggeber

- Letzte Auslieferung bewerten
- Neue Prioritäten gemeinsam festlegen
- Umfang der nächsten Stufe festlegen
- Lektionen austauschen



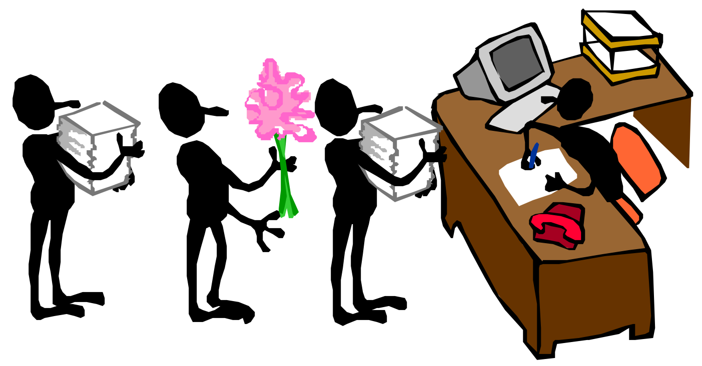
 ©Copyright: Coldewey Consulting 2001; All rights reserved



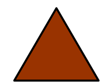


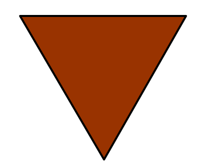
Der Anwender im Team ersetzt die Spezifikation

- Eine Endanwenderin ist vollzeit im Team
- Fachliche Fragen werden sofort mit ihr geklärt
- Sie wird bei allen Entscheidungen über Fachlichkeit oder Oberfläche einbezogen
- Sie arbeitet beim Testen mit



 ©Copyright: Coldewey Consulting 2001; All rights reserved





Minimalismus: Ballast abwerfen

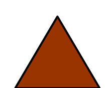
Gerade noch ausreichende Methode

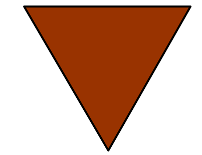
- Überflüssige Regeln kosten viel Geld
- Je größer das Team, um so mehr Regeln braucht das Projekt
- Je höher die Kritikalität um so mehr Zeremonie

Minimales Design

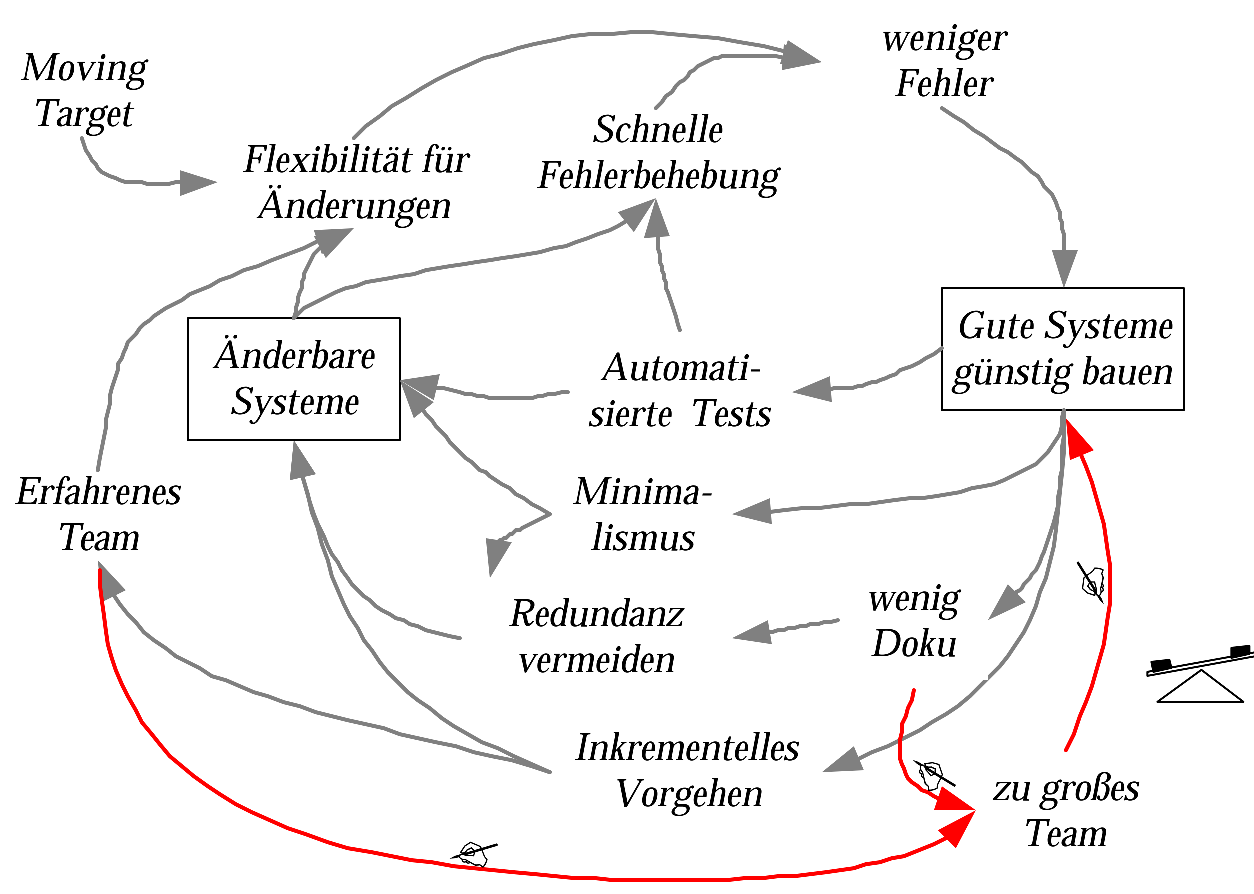
- YAGNI: „You aren‘t gonna need it“
- Features erst einbauen, wenn sie angefordert werden
- Test-first Programmierung

 ©Copyright: Coldewey Consulting 2001; All rights reserved



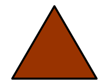


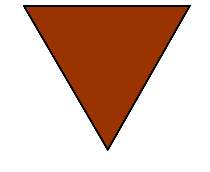
Die Systemsicht zeigt Stärken und Grenzen



Make it right the last time - Jim Highsmith [Hig00]

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Wenige Regeln und viel Interaktion führen zu Überraschungen

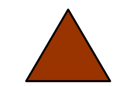
- Komplexe Systeme zeigen häufig überraschendes Verhalten: Wirtschaft, soziale Systeme, Physik, Gehirn
- Dieses Verhalten heißt „Ermergence“ [Hol98]
- Agile Verfahren nutzen dieses Phänomen

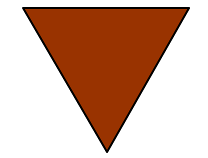
$$? \dot{E} ? 4??$$
$$? ? \overset{?}{E} ? ? \frac{1}{c}\frac{?\overset{?}{B}}{?t}$$
$$? \overset{?}{B} ? 0$$
$$? ? \overset{?}{B} ? \frac{4?}{c}\overset{?}{j} ? \frac{1}{c}\frac{?\overset{?}{E}}{?t}$$

War es ein Gott, der diese Zeilen schrieb?

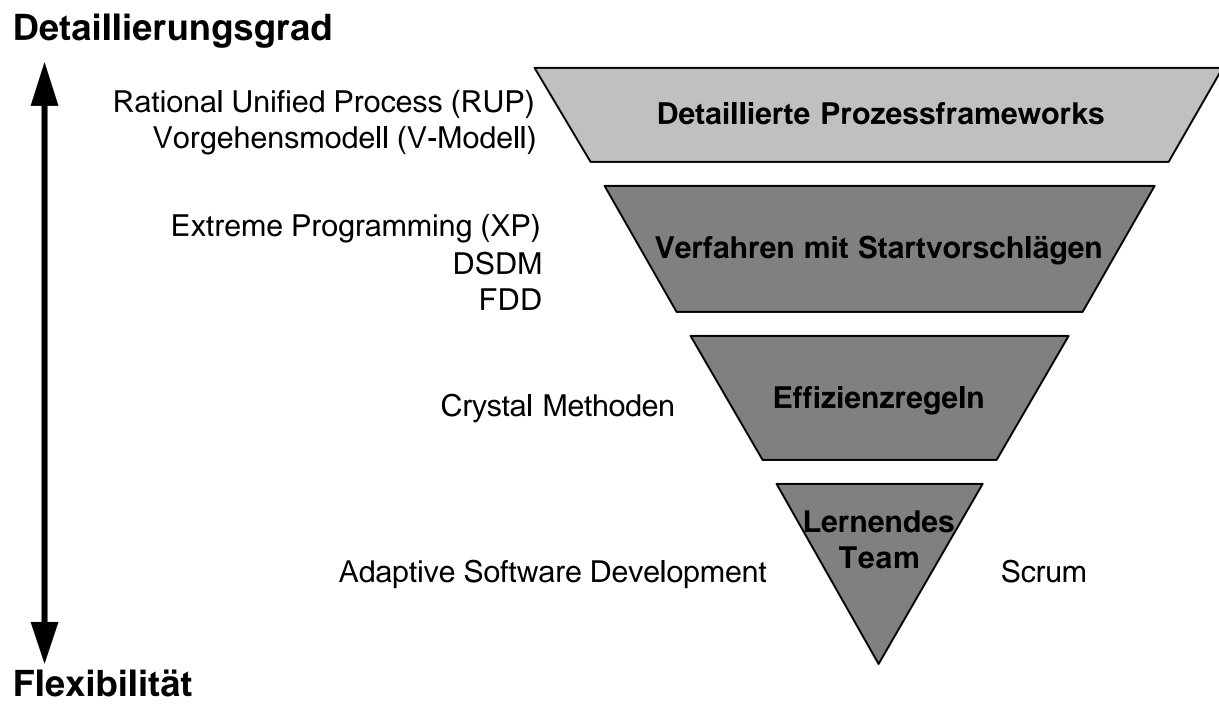
Ludwig Boltzmann

Folie 38; ©Copyright: Coldewey Consulting 2001; All rights reserved

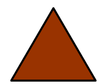


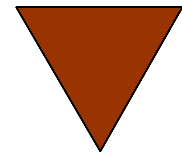


Die einzelnen Verfahren sind auf unterschiedlichen Ebenen



 ©Copyright: Coldewey Consulting 2001; All rights reserved

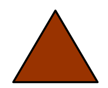


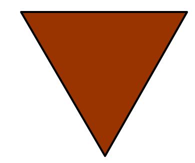


Teil IV:
Drei Beispiele

We need to control the development of software by making many small adjustments, not by making a few large adjustments, kind of like driving a car.
Kent Beck [Bec00]

 ©Copyright: Coldewey Consulting 2001; All rights reserved

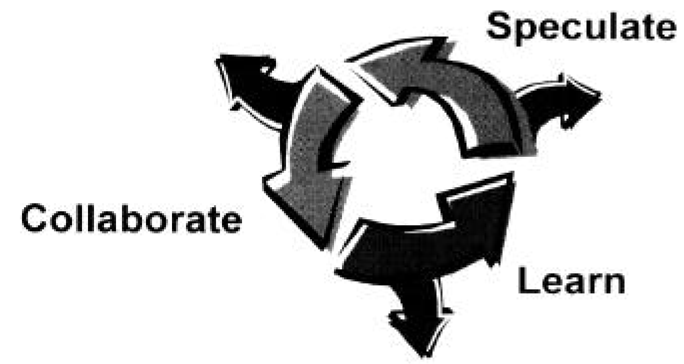




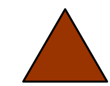
Adaptive Software Development (ASD)
Jim Highsmith

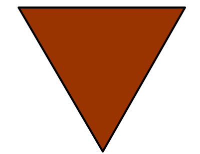
- Projektstart:
  Gemeinsame Vision
- Planung: Eine Iteration durch das Team
- Iterationsdauer: etwa 4 Wochen, Vorgehen vom Team festgelegt
- Inkrementenende: Ausgelieferte Software, Reflektionsmeeting, Verbesserungsliste

Siehe [Hig00]



 ©Copyright: Coldewey Consulting 2001; All rights reserved



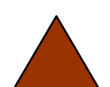


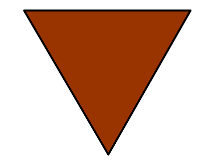
Crystal Methodenfamlie
Alistair Cockburn

Prinzipien:

- Direkte Kommunikation ist der effizienteste Informationsweg
- Je größer das Team, um so mehr Regeln
- Überflüssige Regeln sind teuer
- Hohe Kritikalität braucht mehr Zeremonie
- Formalismus heißt nicht Disziplin, Prozess nicht Fähigkeit, Dokumentation nicht Verständnis
- Mehr Feedback und Kommunikation heißt weniger Zwischenprodukte
- Effizienz ist Dispositionsmasse abseits vom kritischen Pfad

 ©Copyright: Coldewey Consulting 2001; All rights reserved

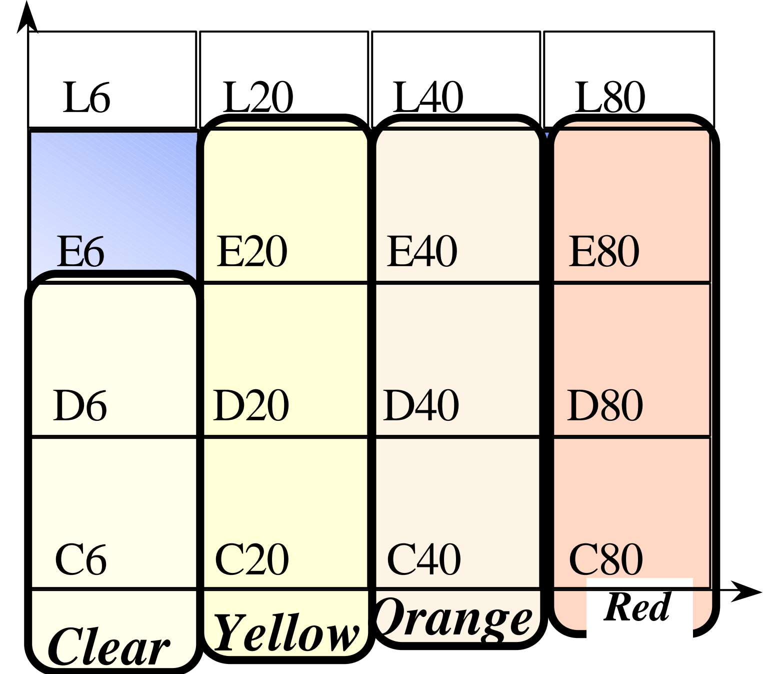




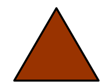
Anhand dieser Prinzipien erstellt das Team selbst den Prozess

- Regelmäßige Prozessworkshops ein- bis zweimal pro Inkrement
- Muster für Methoden unterschiedlicher Anforderungen

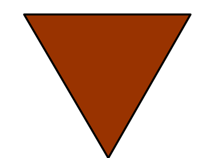
Siehe [Coc02]



Aus: Alistair Cockburn: *Designing a Light Methodology*, [Coc98]



 ©Copyright: Coldewey Consulting 2001; All rights reserved

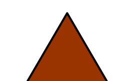


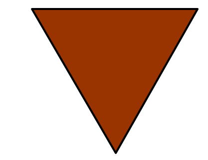
eXtreme Programming
Kent Beck, Ward Cunningham, Ron Jeffries

Grundwerte:

- Unmittelbares Feedback
- Streben nach Einfachheit
- Inkrementelle Weiterentwicklung
- Änderungen willkommen heißen
- Qualitätsarbeit leisten

 ©Copyright: Coldewey Consulting 2001; All rights reserved





eXtreme Programming baut auf zwölf Techniken auf

Metapher

Nachhaltiges Tempo

Kunde vor Ort

Einfaches Design

Akzeptanztests

Refactoring

Fortlaufende Integration

Pair Programming

Kurze Inkremente

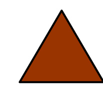
Entwicklertests

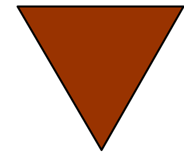
Planungsspiel

Gemeinsame Verantwortung

Programmierstandards

 ©Copyright: Coldewey Consulting 2001; All rights reserved



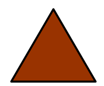


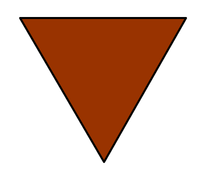
Teil V:
Echo auf agile Entwicklung

Traue keiner Statistik, die Du nicht selbst gefälscht hast

Winston Churchill

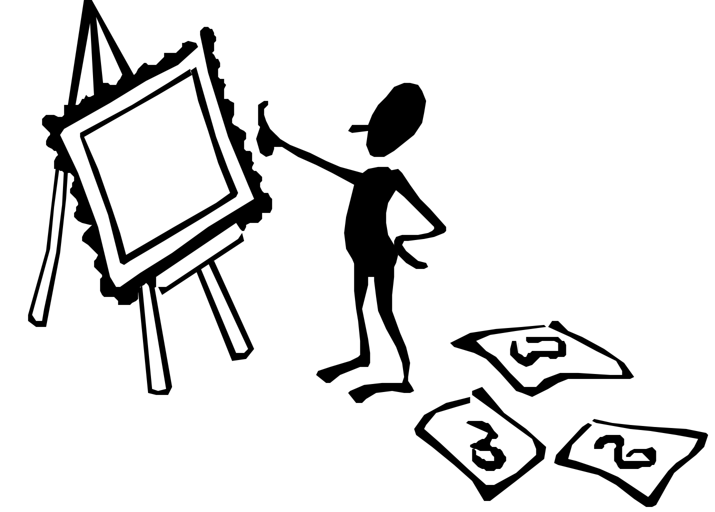
 ©Copyright: Coldewey Consulting 2001; All rights reserved



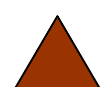


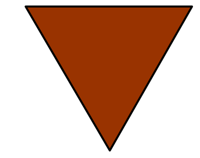
Erste Umfragen zu agiler Entwicklung

- Jährliche Umfrage des Cutter Consortiums
- 159 Unternehmen der IT-Branche zum realen Einsatz aktueller Techniken
- Teilnahme offen und freiwillig -> Nicht repräsentativ!
- Quelle: „Benchmark Review“, 12/2001, Cutter IT Consortium

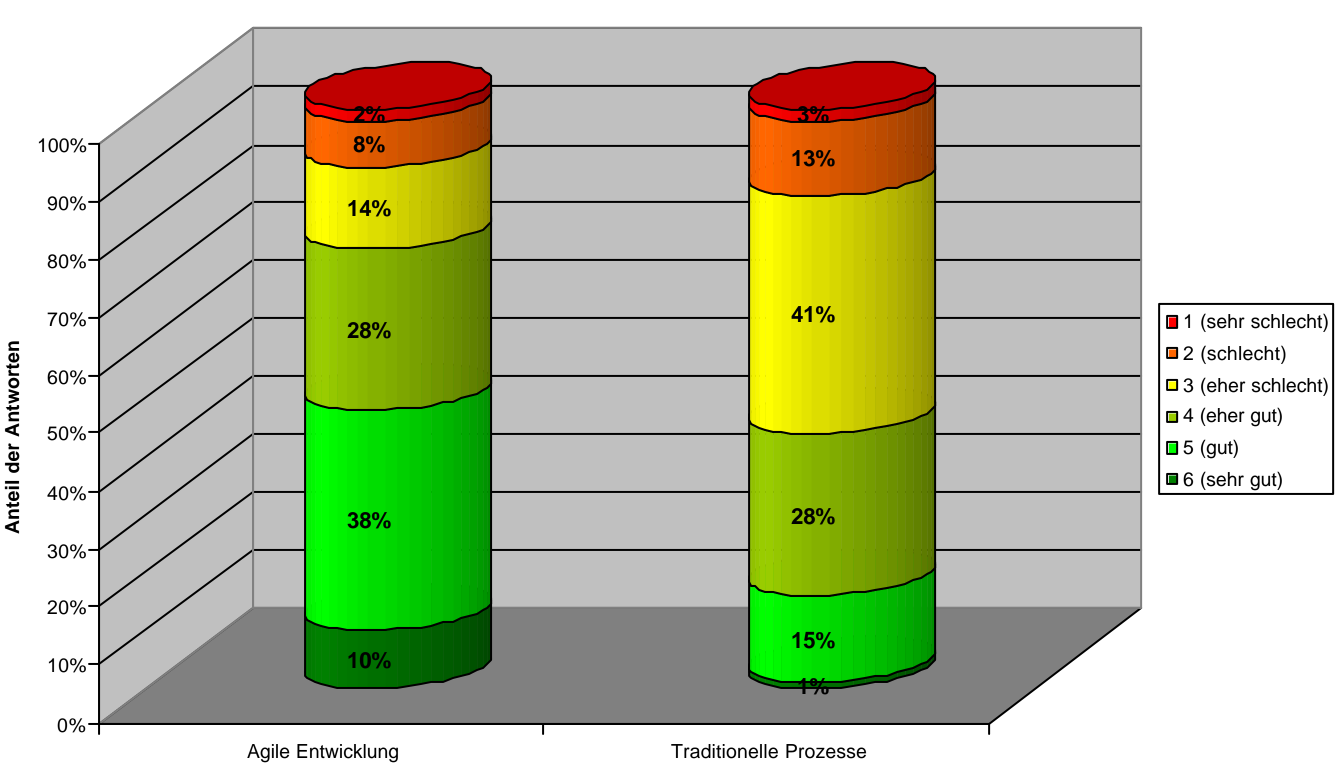


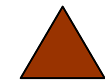
 ©Copyright: Coldewey Consulting 2001; All rights reserved



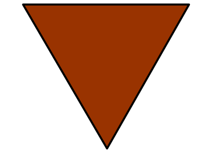


Agile Entwicklung verbesserte die Moral im Team q.e.e

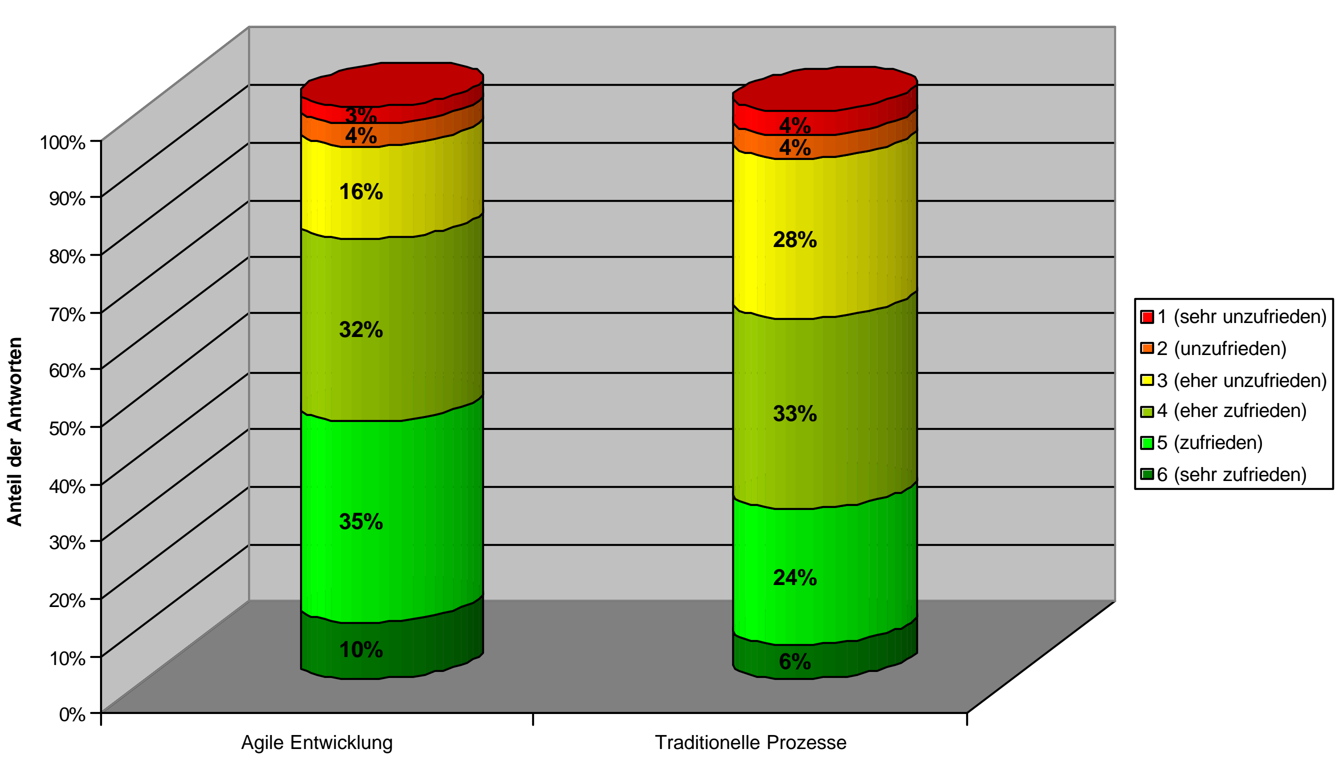




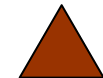
Folie 48; ©Copyright: Coldewey Consulting 2001; All rights reserved

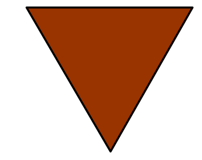


Agile Entwicklung erhöhte die Kundenzufriedenheit q.e.e

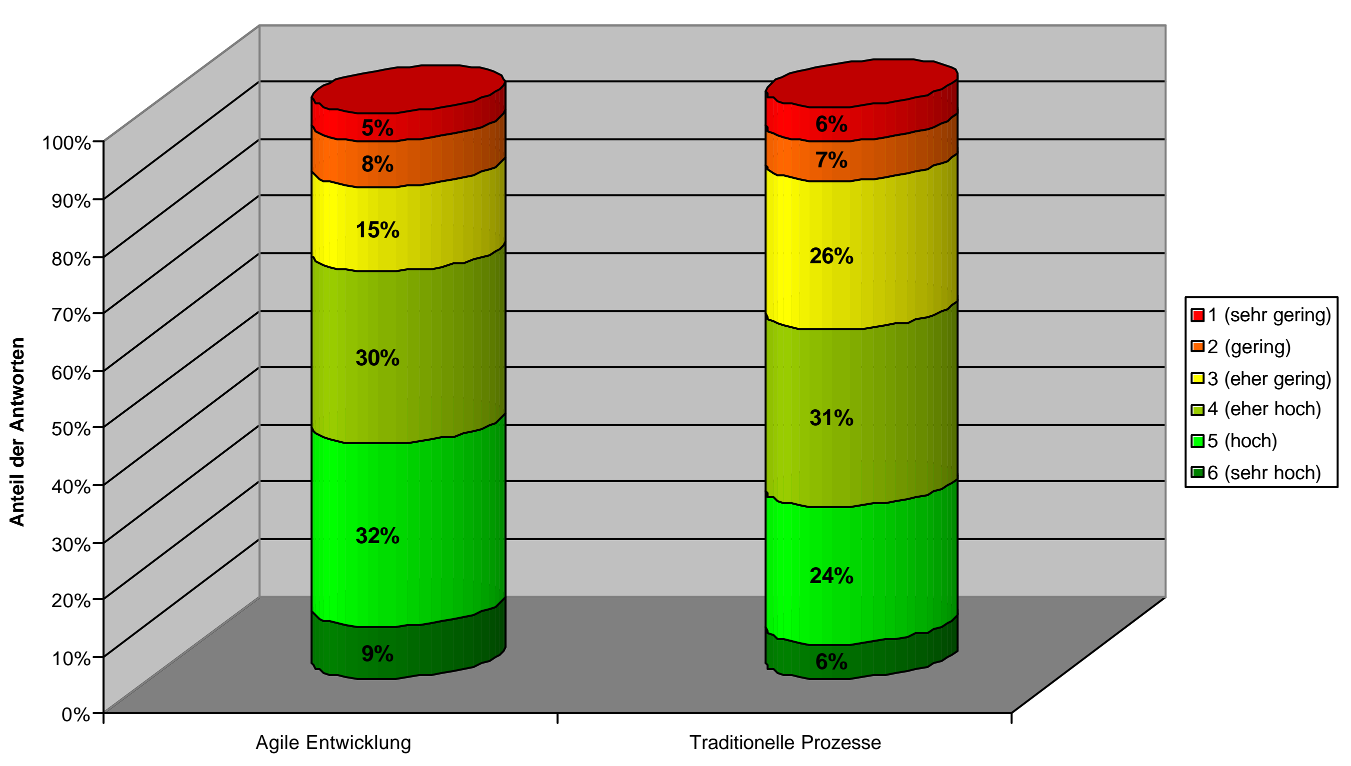


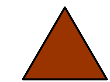
Folie 49; ©Copyright: Coldewey Consulting 2001; All rights reserved



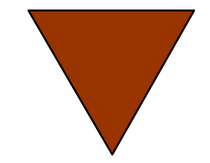


Agile Entwicklung erhöhte die Chancen eines guten Return of Investment q.e.e





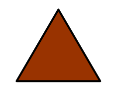
 ©Copyright: Coldewey Consulting 2001; All rights reserved

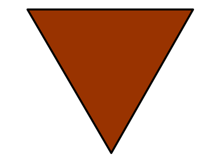


Zusammenfassung

- Agile Entwicklung bildet in vielen Bereichen eine Alternative zu traditionellen Prozessen
- Sie erhöht die Flexibilität auf Kosten der langfristigen Planbarkeit
- Eine Reihe eng abgestimmter Techniken sollen Konvergenz und Qualität sichern
- Erfahrungen mit kleinen Teams sind sehr gut. Große Teams müssen modifizierte Ansätze verwenden

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Warum wurden diese Ideen nicht schon in den Siebzigern verfolgt?

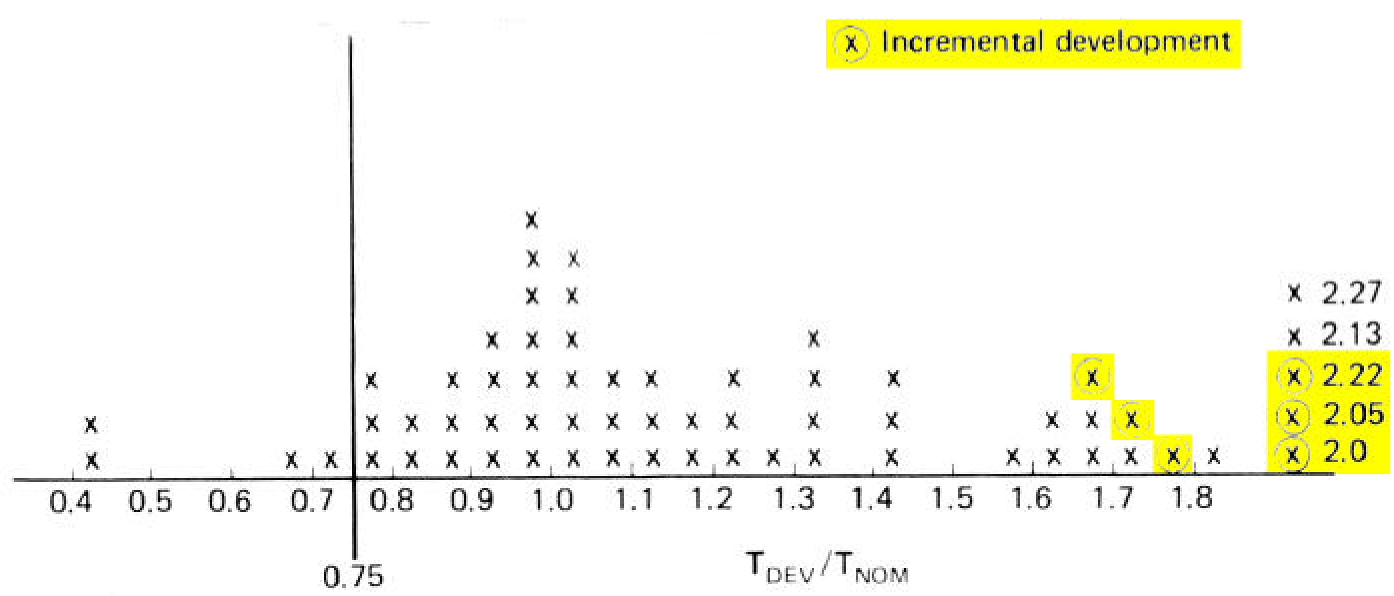
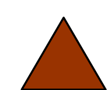


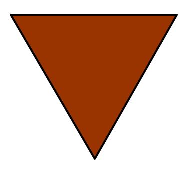
FIGURE 27–9 Schedule compression or expension in COCOMO data base projects

Aus: Barry W. Boehm: *Software Engineering Economics* [Boe81], Seite 471

? Es fehlte an den Werkzeugen, um inkrementell zu entwickeln

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Referenzen

[Bec00] Kent Beck: Extreme Programming Explained - Embrace Changes; Addison-Wesley, Reading, Massachusetts, 2000; ISBN 0-201-61641-6

[Boe81] Barry W. Boehm: Software Engineering Economics; Prentice Hall, Eaglewood Cliffs, New Jersey, 1981; ISBN 0-13-822122-7

[Coc98] Alistair Cockburn: Designing a light Methodology; Tutorial erhältlich bei: http://members.aol.com/humansandt/

[Coc02] Alistair Cockburn: Agile Software Development; Addison-Wesley, Reading, Massachusetts, 2002; ISBN 0-201-69969-9

[Fow99] Martin Fowler: Refactoring - Improving the Design of Existing Code; Addison-Wesley, Reading, Massachusetts, 1999; ISBN 0-201-48567-2

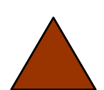
[Hig00] James A. Highsmith III: Adaptive Software Development - A Collaborative Approach to Managing Complex Systems; Dorset House, New York, 2000; ISBN 0-932633-40-4

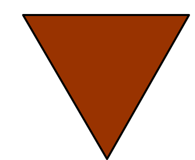
[Hol98] John H. Holland; Emergence - From Chaos to Order; Oxford University Press, New York, Oxford, 1998; ISBN 0-19-850409-8

[Sen90] Peter Senge: The Fifth Discipline - The Art & Practice of The Learning Organization; Currency Doubleday, New York, 1990; ISBN 0-385-26095-4

Der Drache und sein Bezwinger wurden abgewandelt vom Buchumschlag von: Alfred Aho, Ravi Sethi, Jeffrey Ullman: Compilers - Principles, Techniques, and Tools; Addison-Wesley, Reading, Massachusetts, 1986; ISBN 0-201-10088-6

 ©Copyright: Coldewey Consulting 2001; All rights reserved





Weitere Links zum Thema

- Dieser Vortrag: http://www.coldewey.com
- Agile Entwicklung: http://www.agileprozesse.de/

Homepages der Verfahren

- ASD: http://http.adaptivesd.com
- Crystal: http://www.crystalmethodologies.org/
- eXtreme Programming: http://www.xprogramming.com/

Weitere Verfahren

- Scrum:http://www.controlchaos.com
- DSDM: http://www.dsdm.com
- FDD: http://www.nebulon.com

 ©Copyright: Coldewey Consulting 2001; All rights reserved