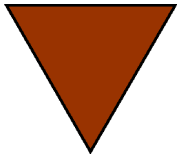
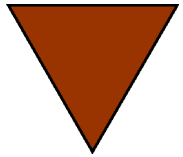


Agile Prozesse - Eine Einführung

Glashütten, den 7. März 2001

Jens Coldewey
Coldewey Consulting
Curd-Jürgens-Str. 4
D-81739 München
jens_coldewey@acm.org



Teil I:
Schwarze Schafe in Schottland

We must challenge our most fundamental assumptions about software development

Jim Highsmith [Hig00]

Schwarze Schafe in Schottland: Der flüchtige Blick

proximately one-third of large software projects are never completed. Much of the enormous loss from these aborted projects can be attributed to poor requirements definition.

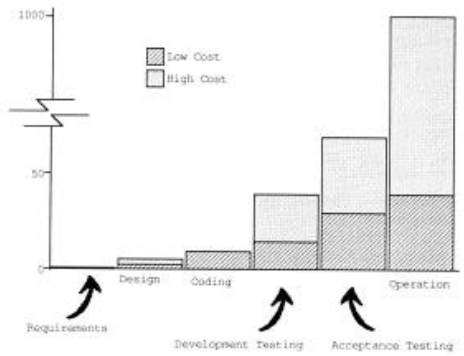


Figure 2-4. Boehm's observations on project cost.

The situation looks even worse when we consider the catastrophes resulting from incorrect design decisions based on early false assumptions. For example, on the Ford Pinto manufactured in the 1970s, the position of the fuel tank mounting bolts

Aus: Donald Gause, Gerald Weinberg: *Exploring Requirements - Quality Before Design* [GaW89], Seite 18

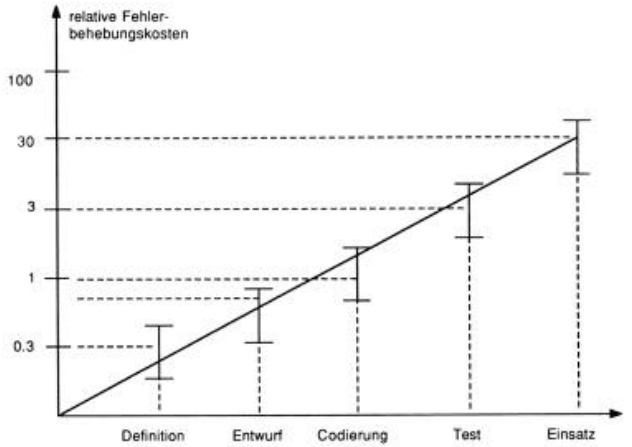


Abb. 1.4 Relative Fehlerbehebungskosten nach Boehm

Wir starten den Software-Entwicklungsprozeß mit ganz bestimmten Anforderungen, wie beispielsweise der Anforderung nach einer vorgegebenen Produktqualität, oder der Anforderung, das Projekt termingerecht und kostengerecht, d. h. im Budgetrahmen,

Aus: Ernest Wallmüller: Software Qualitäts-sicherung in der Praxis [Wal90], Seite 13

Die Kosten von Fehlern in Software steigen exponentiell mit der Zeit

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

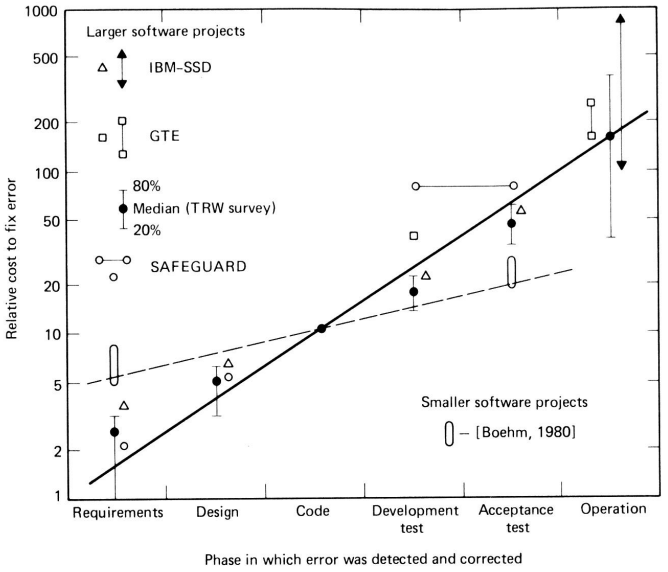
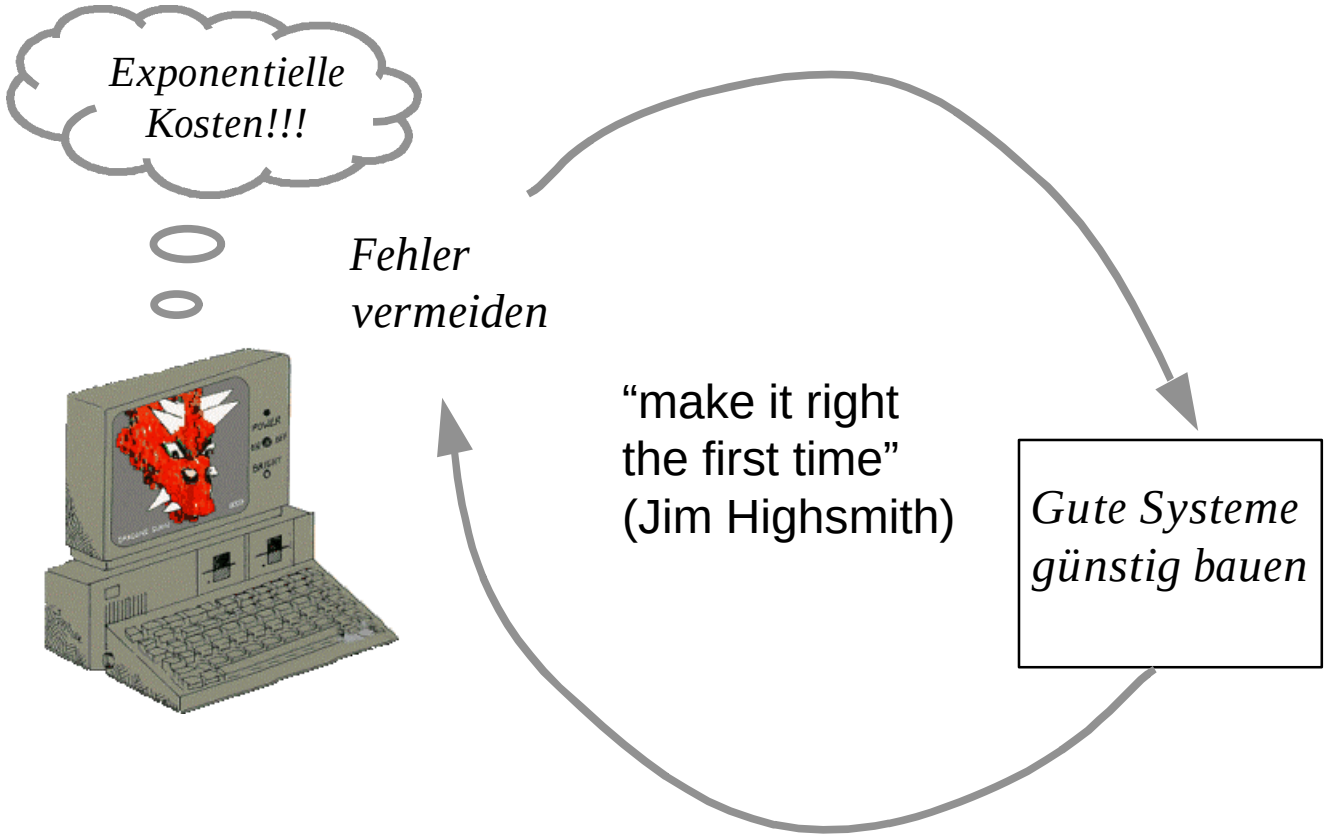


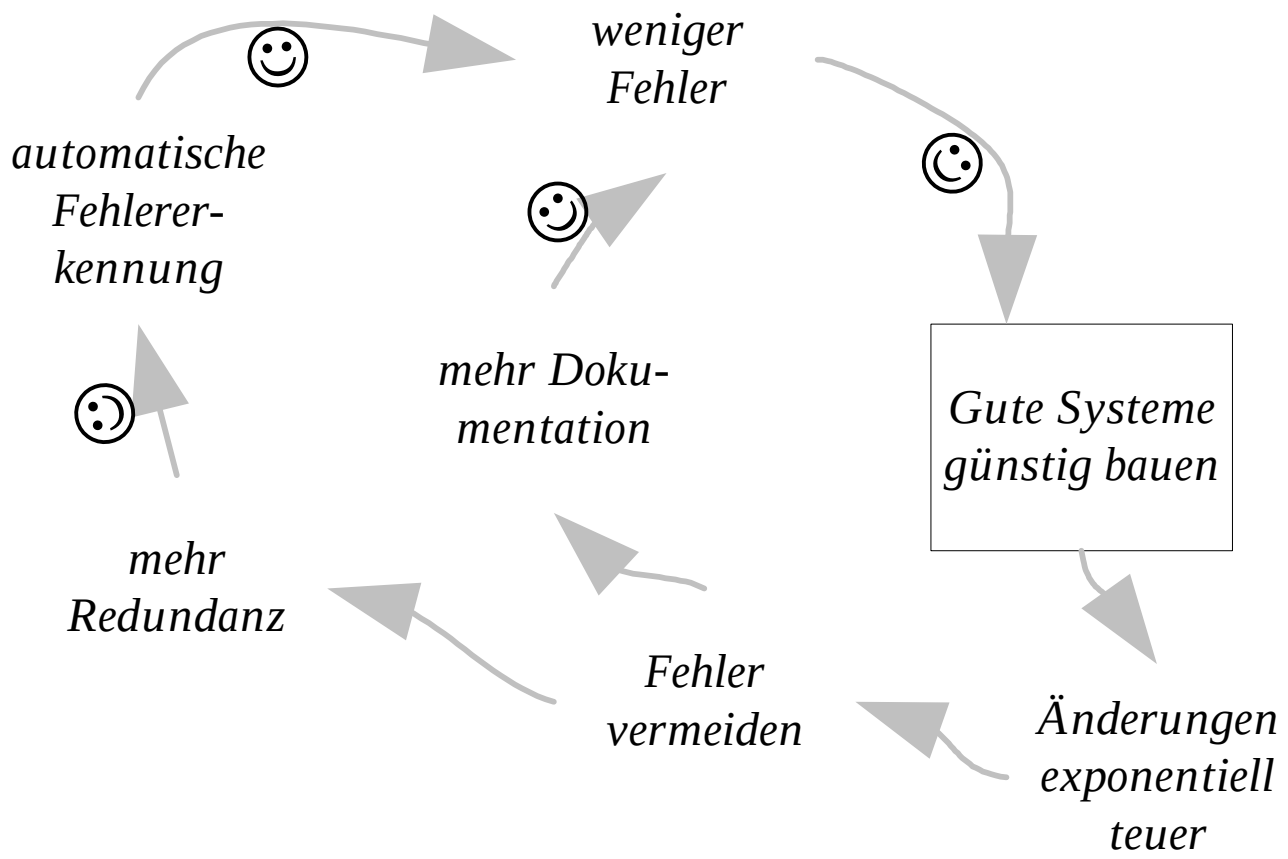
FIGURE 4-2 Increase in cost-to-fix or change software throughout life-cycle

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

Ein logisches Gegenmittel scheint, Fehler zu vermeiden



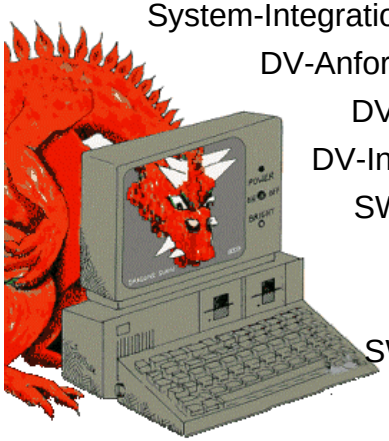
Klassische Prozesse erreichen dies Ziel durch Dokumentation und Redundanz



Ein genauerer Blick auf das Schaf (I): 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“

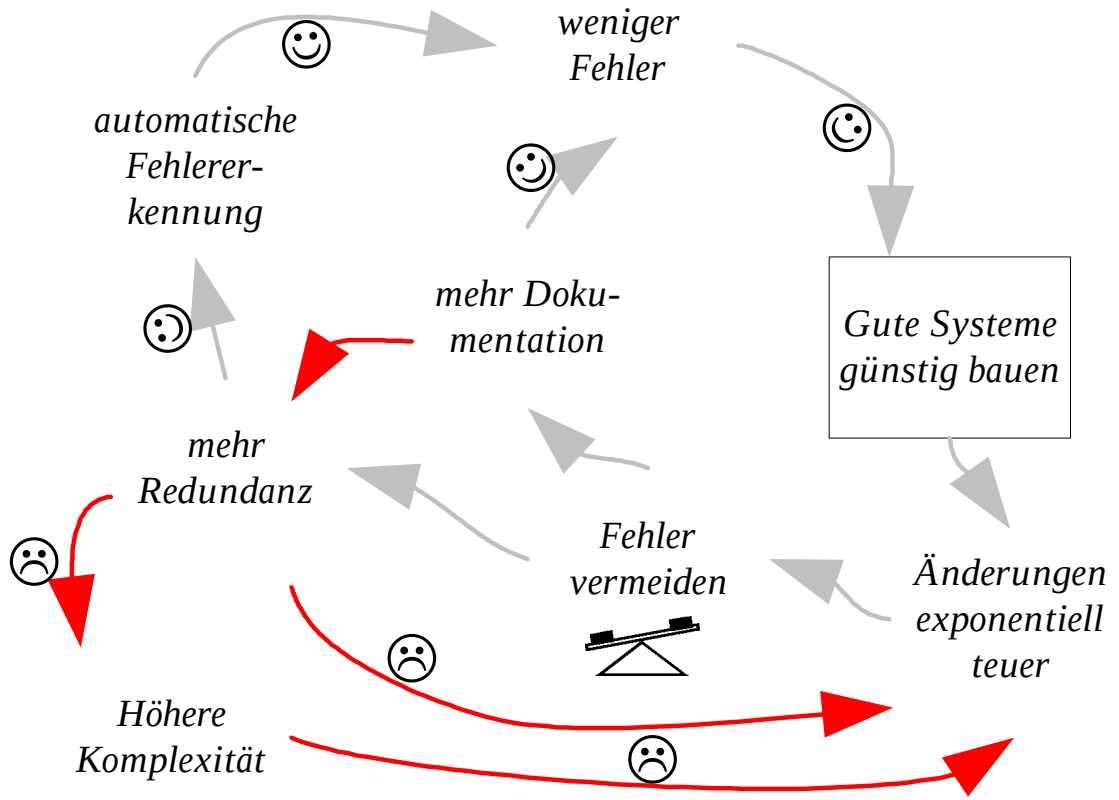
Aus: Barry W. Boehm: *Software Engineering Economics* [Boe81], Seite 39f



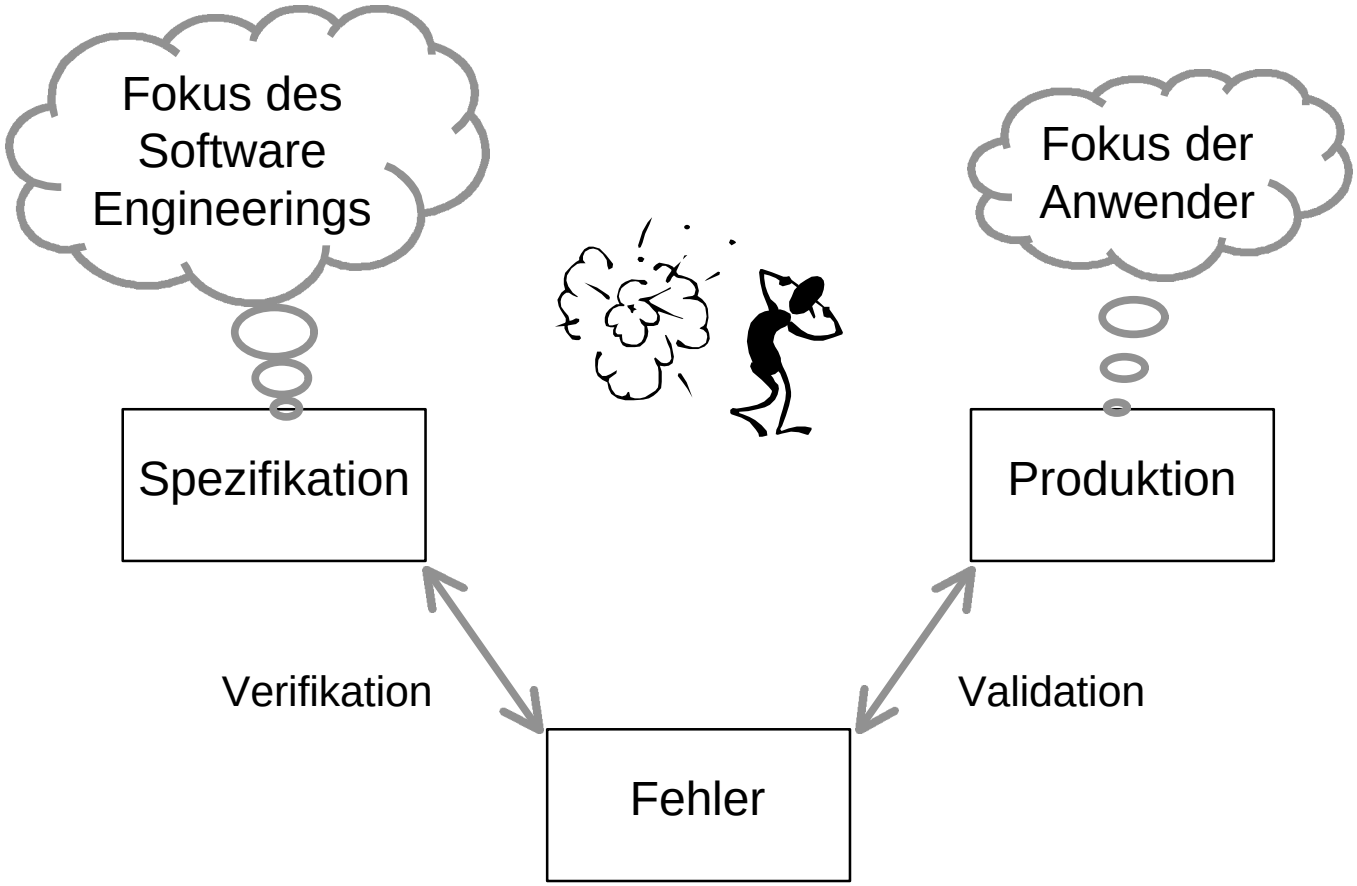
Systemanforderungen
Systemarchitektur
System-Integrationsplan
DV-Anforderungen
DV-Architektur
DV-Integrationsplan
SW-Anforderungen
SW-Architektur
Schnittstellenentwurf
SWKE-Integrationsplan
Datenkatalog
SW-Entwurf
Implementierungsdok. (Modul)
Implementierungsdok. (Komponente)
Implementierungsdok. (SWKE)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

Redundanz

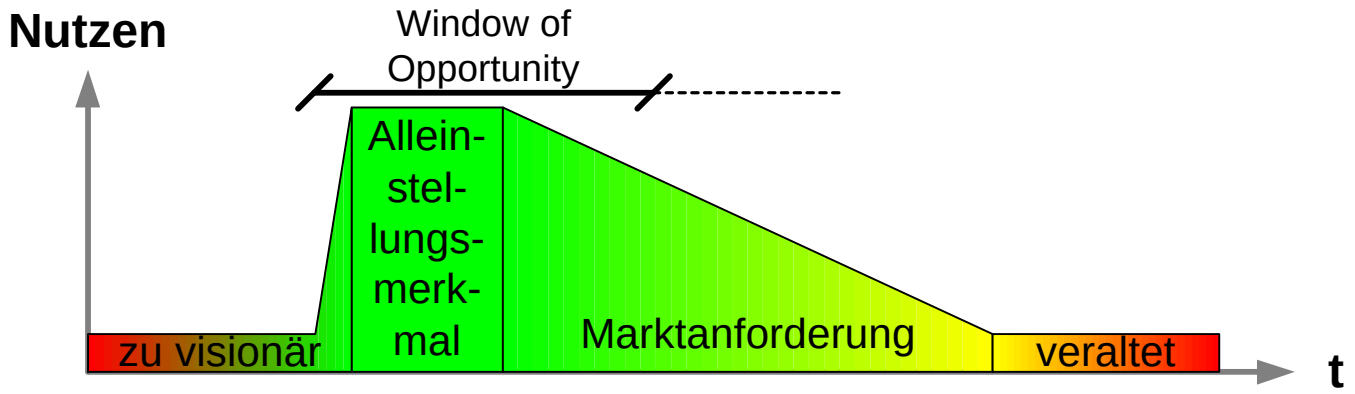
Offensichtlich muss die Systemsicht noch ergänzt werden



Ein genauerer Blick auf das Schaf (II): Was sind eigentlich Fehler?

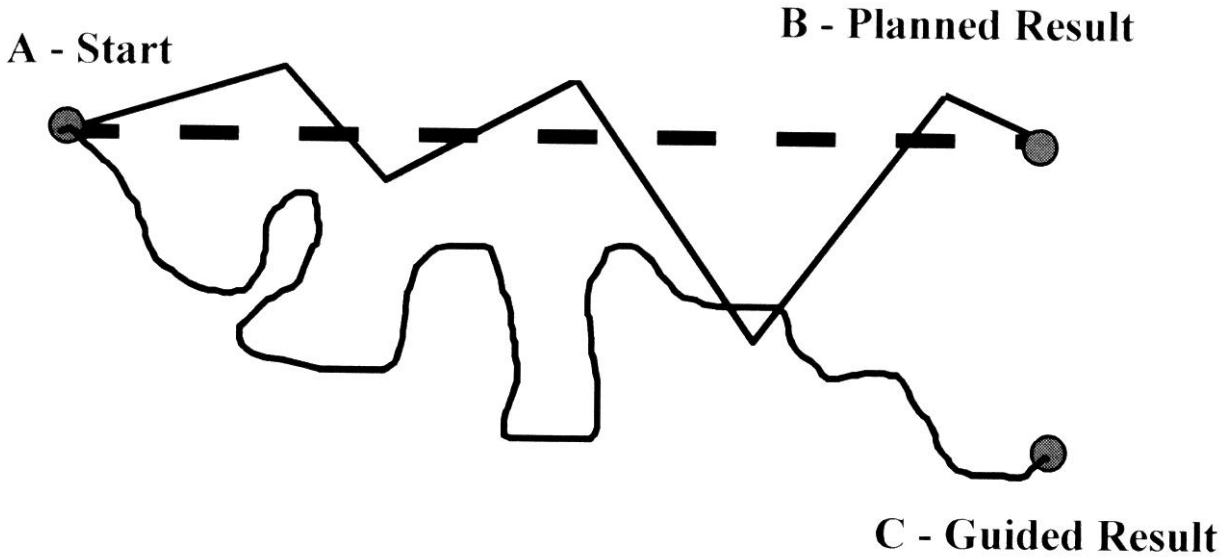


Betriebswirtschaftlich ist ein System nur in einer bestimmten Zeitspanne korrekt



- 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

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]

Projekte haben sich in den letzten 25 Jahren grundlegend geändert

Typische Projekte der Siebziger

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

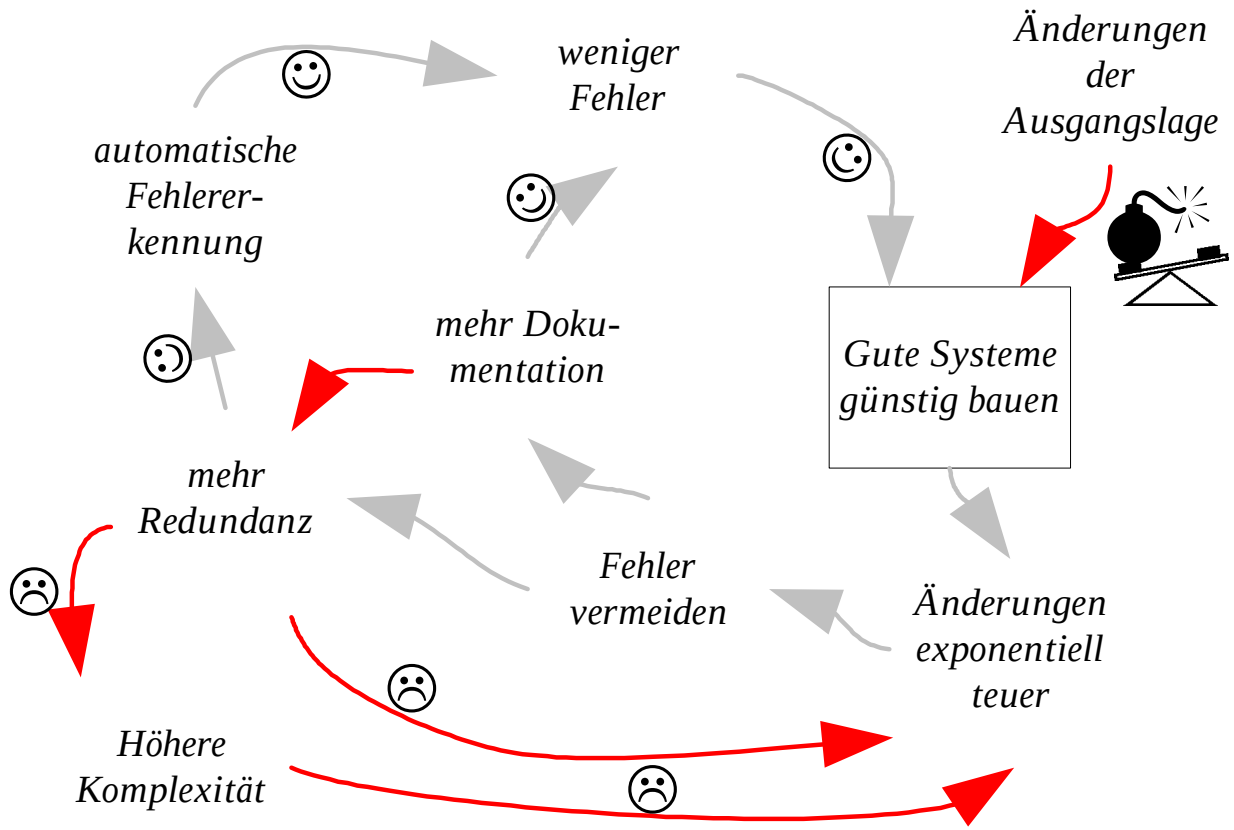
⇒ „Moving Target“ war Zeichen schlechten Managements

Typische Projekte heute

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

⇒ „Moving Target“ ist die Regel

Die Systemsicht zeigt das Problem



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?

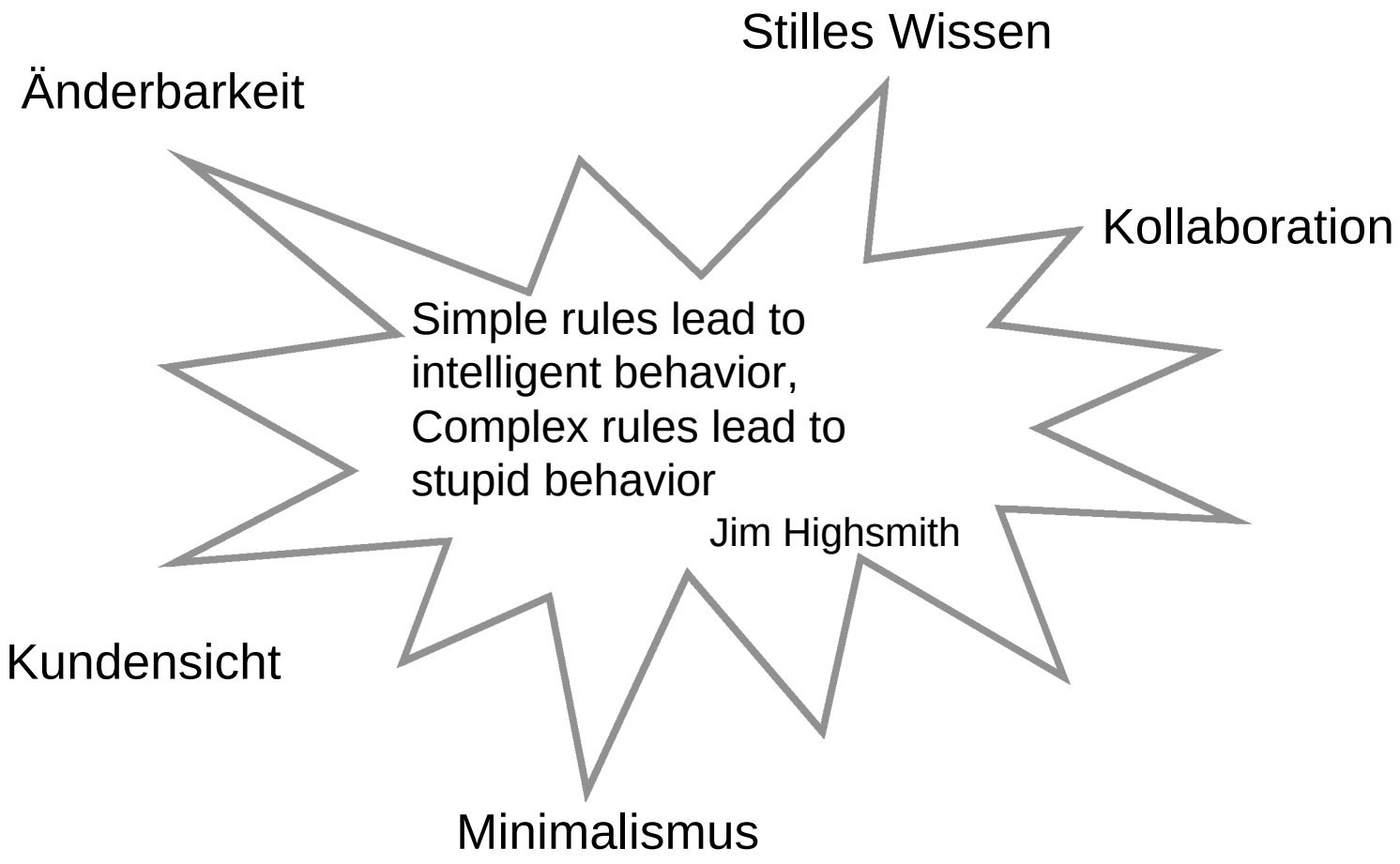
⇒ Wie sieht die Entwicklung aus, wenn sie auf Änderbarkeit ausgerichtet ist?

Teil II:
Über sieben Brücken musst Du geh‘n

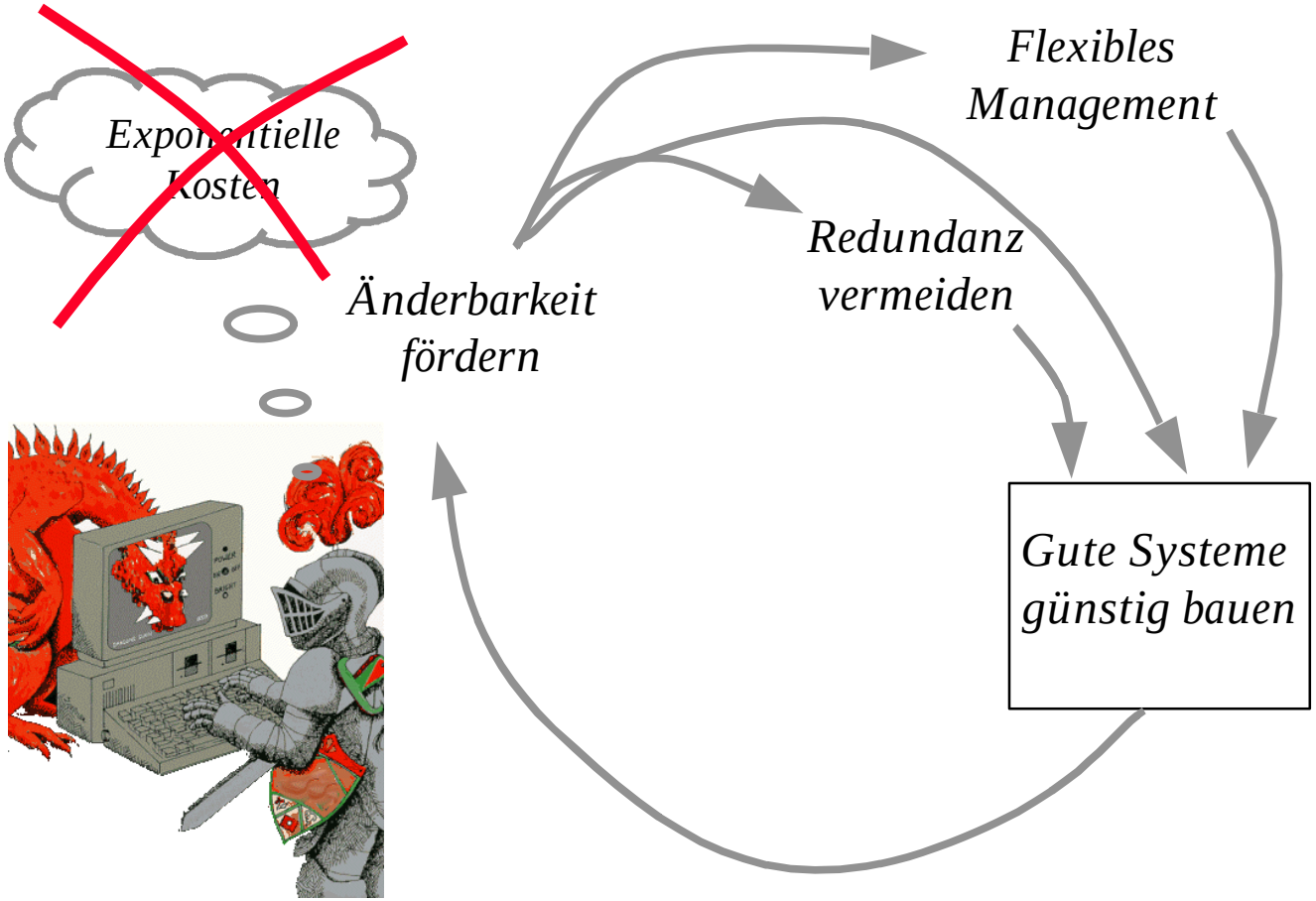
Our organizations work the way they work, ultimately, because of how we think and how we interact. Only by changing how we think we can change deeply embedded policies and practices

Peter Senge [Sen90]

Die Grundprinzipien Agiler Prozesse



Wie macht man änderbare Systeme?



Flexible Projekte brauchen flexibles Management

„Command-Control“

- Ausgabe klarer Anweisungen an Einzelne
- Vorgabe des Arbeitsverfahrens
- Kontrolle der Ergebnisse

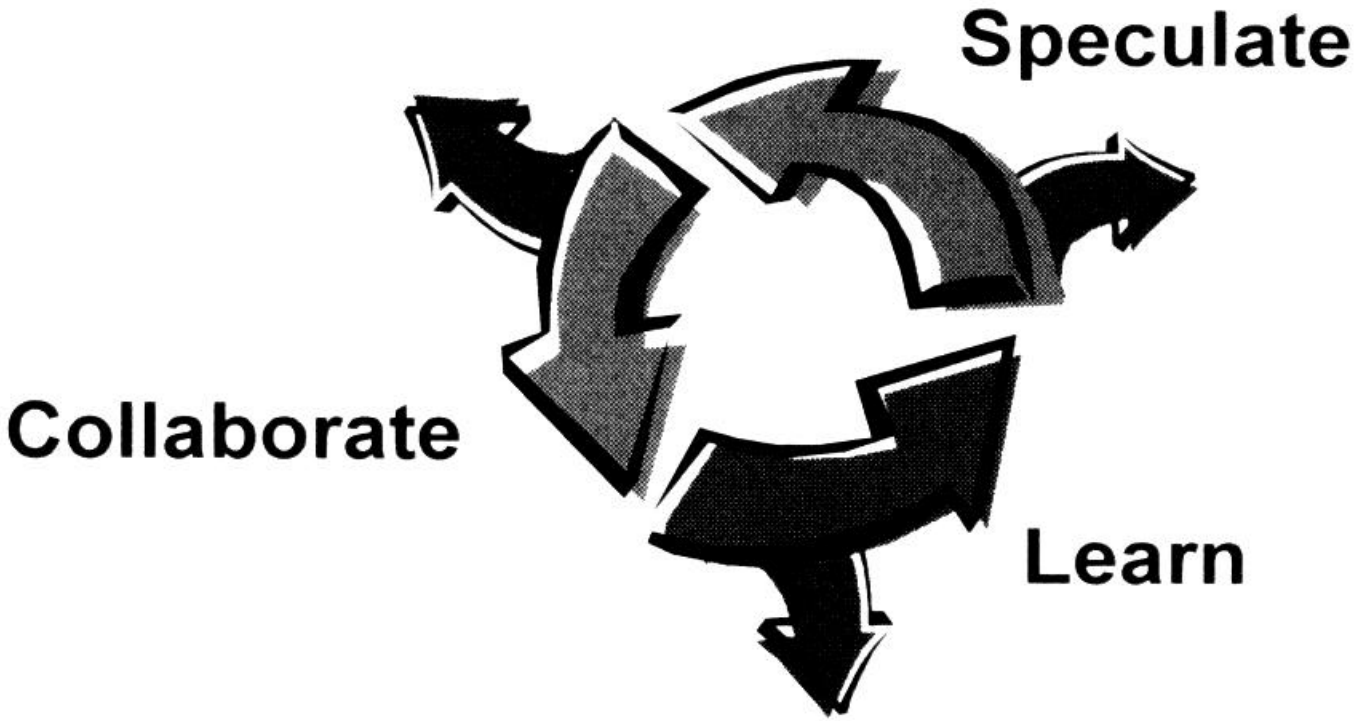
⇒ Abweichungen vom Plan sind Zeichen schlechter Arbeit

Flexibles Management

- Einschwören auf gemeinsames Ziel
- Gemeinsame Planung der Arbeit
- Konzentration auf Ergebnisse statt auf Verfahren

⇒ Abweichungen vom Plan sind Zeichen weiterer Erkenntnis

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

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

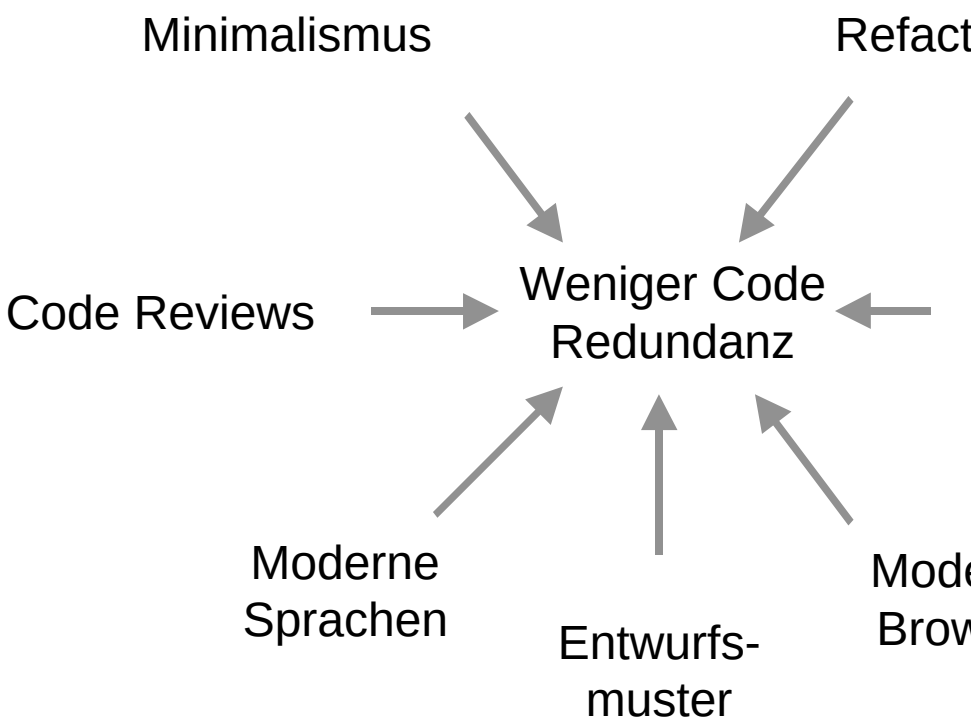
Redundanz vermeiden, bedeutet Konzentration auf das Wesentliche

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

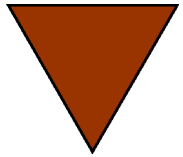
Redundanz

primär	sekundär
Code	
Anwenderdoku	
	Tests
	Verständnis der Domäne
	Verständnis des Systems

Code-Redundanz wird mit Werkzeugen entschärft und mit Design vermieden



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

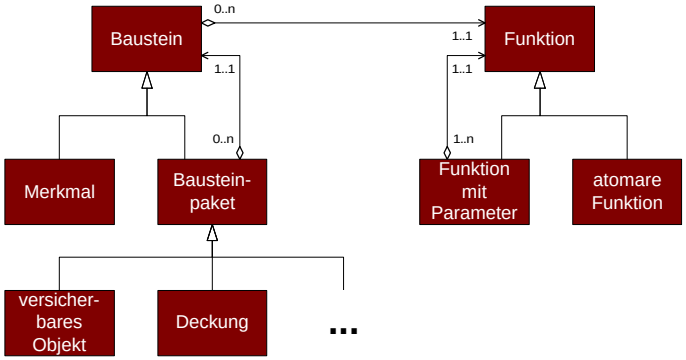


Regressionsfähige Tests stellen Änderbarkeit sicher

- Parallel zum Code werden automatisch ablaufende Testsuiten gebaut
- Fehler sind primär eine Lücke in den Tests, die behoben werden muss
- 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

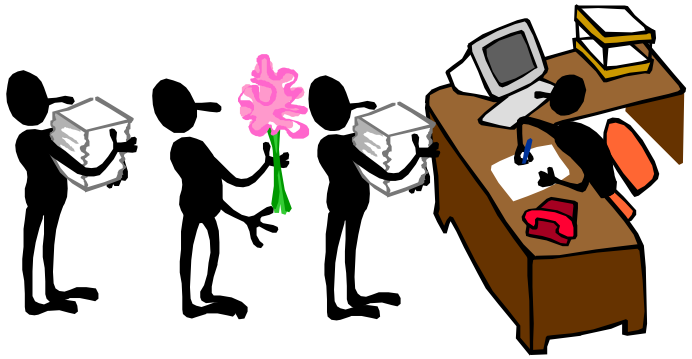
Eine gemeinsame Vision dient der fachlichen und technischen Konvergenz

- 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



Kundenfokus: 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 steuert die fachliche Priorisierung bei



Stilles Wissen: Menschen sind nicht austauschbar

- Mehrwert hinterfragen: Was alle im Team wissen, muss nicht (immer) aufgeschrieben werden
- Manchmal bringt ein kurzes Gespräch mehr als hundert Seiten
- Missverständnisse und Fehler können toleriert werden
- Erfahrung muss weitergegeben werden
- Eingespielte Teams können sehr effizient kommunizieren

Wie Wissen transferiert wird, muss im Einzelfall entschieden werden

Schriftliche Dokumentation

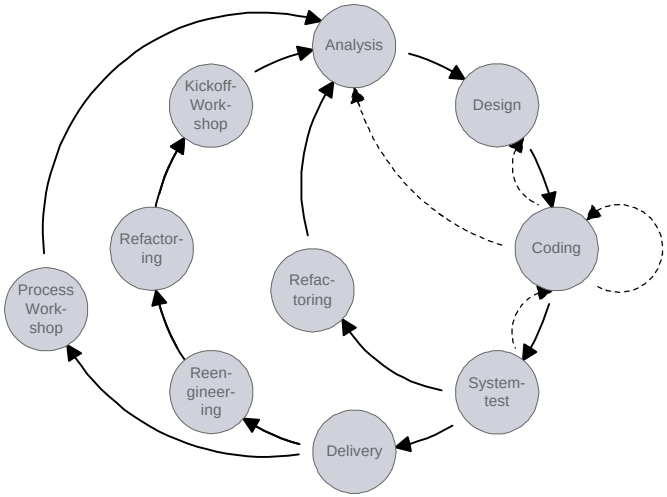
+ Reproduzierbar
+ Kontrollierbar
+ Hohe Streuung
☹ Vieles ist nicht dokumentierbar
☹ Keine Rückkopplung
☹ Schmalbandig
☹ Aufwändig
☹ Redundanz
☹ Scheinqualität (in↔mm)

Kollaboration

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

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



Kollaboration ersetzt hierarchische Strukturen

Software development is a cooperative game, in which people use markers and props to inform, remind and inspire themselves and each other in getting to the next move in the game. The endpoint of the game is an operating software system; the residue of the game is a set of markers to inform and assist the players of the next game. The next game is the alteration or replacement of the system, or creation of a neighboring system.

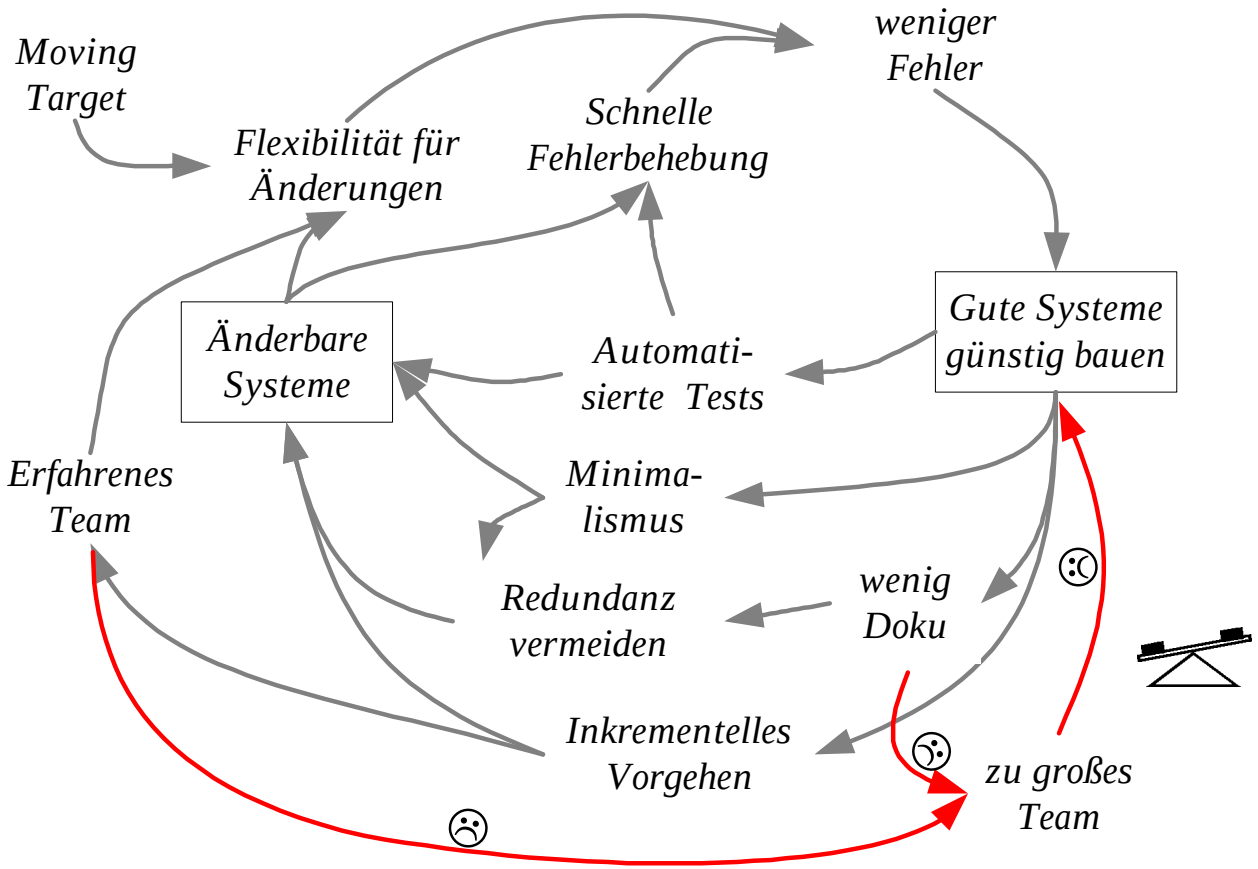
Alistair Cockburn

Minimalismus verhindert das Laufen in die falsche Richtung

„Do the simplest thing that might possibly work“
Kent Beck

- Viele Projekte leiden darunter, dass das falsche Problem gelöst wird („SNARC“)
- Es wird zu viel für die Zukunft gebaut, was doch nie gebraucht wird
- Softwaretechnik scheint wichtiger als Kundennutzen
- Kurze Inkremente lassen keine Spielereien zu
- Man erspart sich viel Ballast im Design, wenn man nur für das aktuelle Problem arbeitet - Ausbauen kann man später
- Die Ergebnisse sind häufig überraschend

Die Systemsicht zeigt Stärken und Grenzen



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

Die Zutaten für ein agiles Projekt

Das Team:

- Erfahrene Entwickler (>50%)
- Geschult in Teamarbeit und Kommunikation
- Ein Fachexperte
- Insgesamt <10 Mitglieder

Die Werkzeuge:

- Moderne Entwicklungs-umgebung
- Sprache mit Abstraktionsvermögen
- Leistungsfähiges Konfigmanagement
- Gute Browser
- Whiteboards

Vertrauen des Managements !!!

Warum wurden diese Ideen nicht schon in den Siebzigern verfolgt?

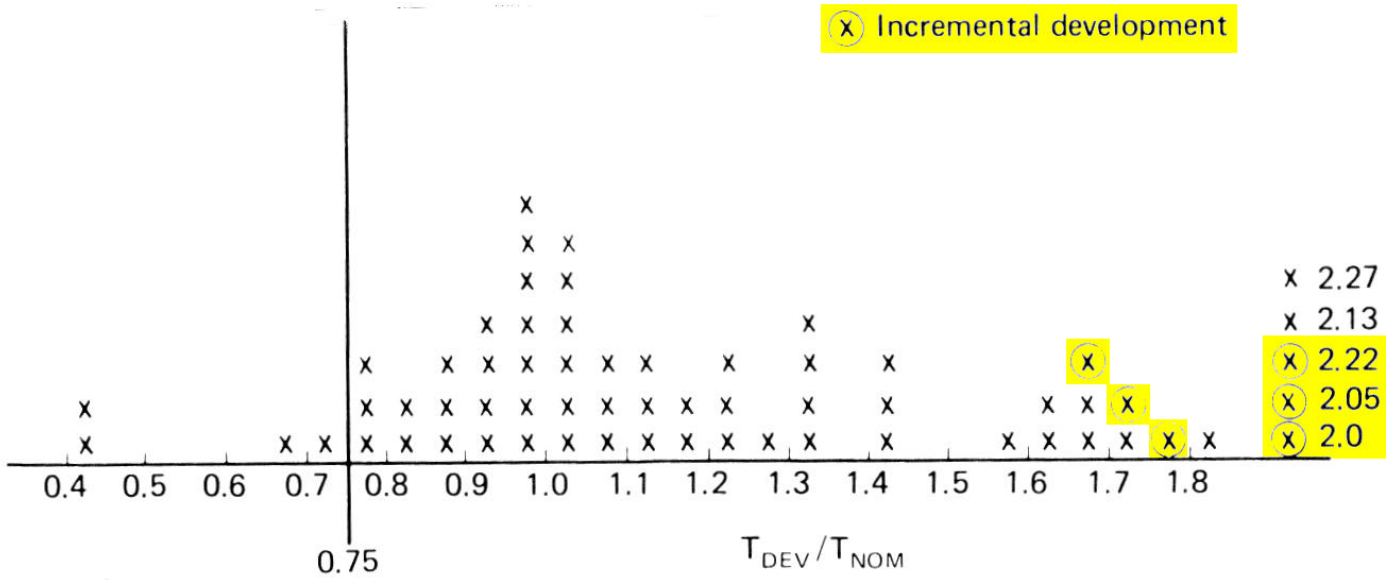


FIGURE 27–9 Schedule compression or expension in COCOMO data base projects

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

⇒ Es fehlte an den Werkzeugen, um inkrementell zu entwickeln

Teil III:
eXtreme Programming

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. This means that we will need the feedback to know when we are a little off, we will need many opportunities to make corrections, and we will have to be able to make those corrections at reasonable cost.

Kent Beck [Bec00]

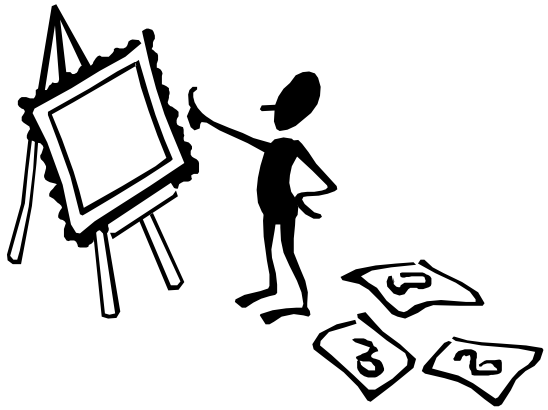
eXtreme Programming baut auf zwölf Techniken auf

- Planungsspiel
- Sehr kurze Inkremente
- Metapher
- Simples Design
- Automatische Tests
- Refaktorieren
- Pair Programming
- Collective Ownership
- Ständige Integration
- 40-Stunden Woche
- Anwender im Projekt
- Programmierstandards

Refaktorieren bedeutet Verbesserung des Designs am laufenden System

- Eine neue Anforderung sollte einfach einbaubar sein
- Ist sie das nicht, so wird das Design erst so umgestellt, dass es einfach wird
- Die Designänderung erfolgt in kleinen, Schritten
- Kein Schritt ändert die Funktionalität des Systems (Testfälle!)

- Das Design ist immer das einfachste Design bezogen auf die implementierten Anforderungen



Programmiert wird immer zu zweit

Nummer 1 sitzt an der Tastatur

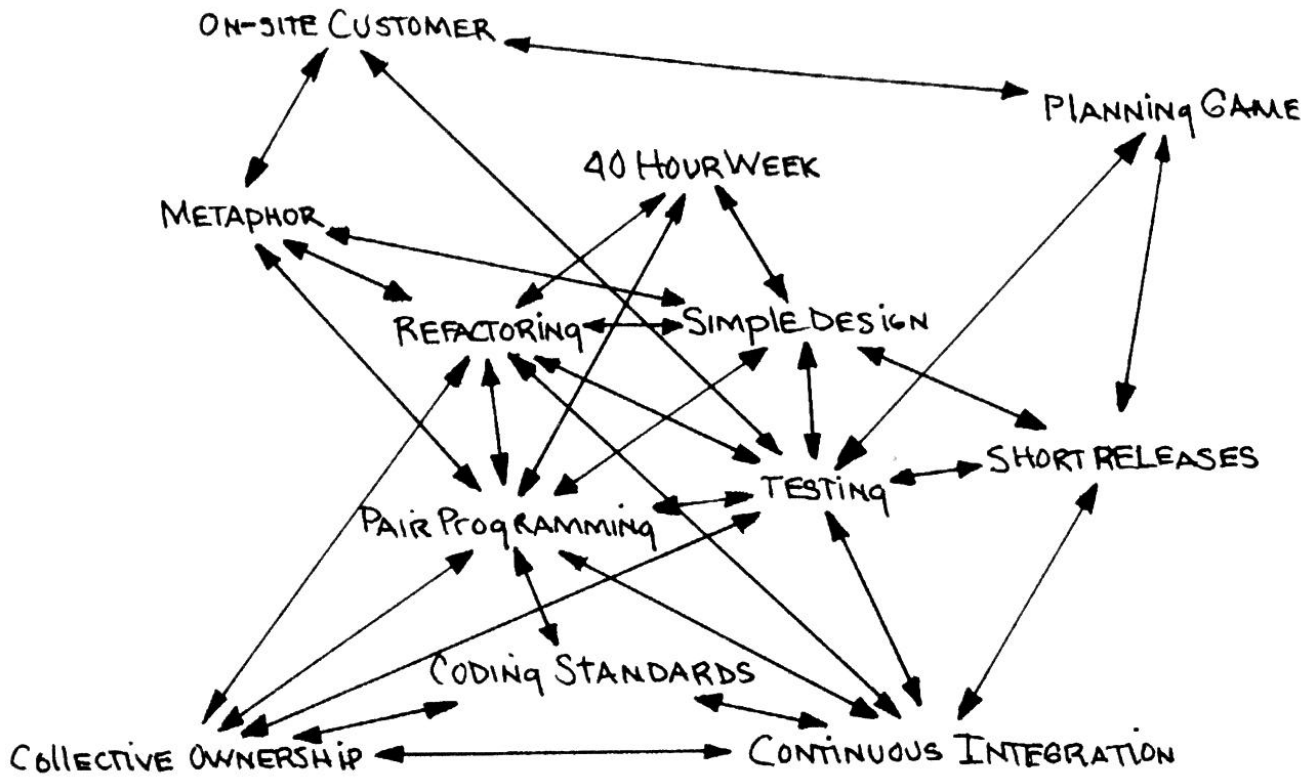
- Schreibt Testfälle für neue Anforderungen
- Verändert das Design so, dass auch die neuen Testfälle laufen

Nummer 2 überlegt:

- Ist der Ansatz richtig?
- Welche Testfälle fehlen noch?
- Lassen sich Design und Code vereinfachen?
- Ist der Code verständlich?

Die Rollen wechseln („Let me drive“)

Alle diese Praktiken bilden ein enges Geflecht



Aus: Kent Beck: *Extreme Programming Explained - Embrace Changes* [Bec00], Seite 61

Teil IV: Crystal und Adaptive Software Development

Gotta watch out for those ski runs at Alta. After shooting up and down them for most of an afternoon, Jim and Alistair got tired of just agreeing with each other all the time and decided to start working together to move the idea of Self-Adapting-Methodologies forward.

Jim Highsmith [Hig01]

Beide Ansätze haben starke Ähnlichkeiten

Crystal:

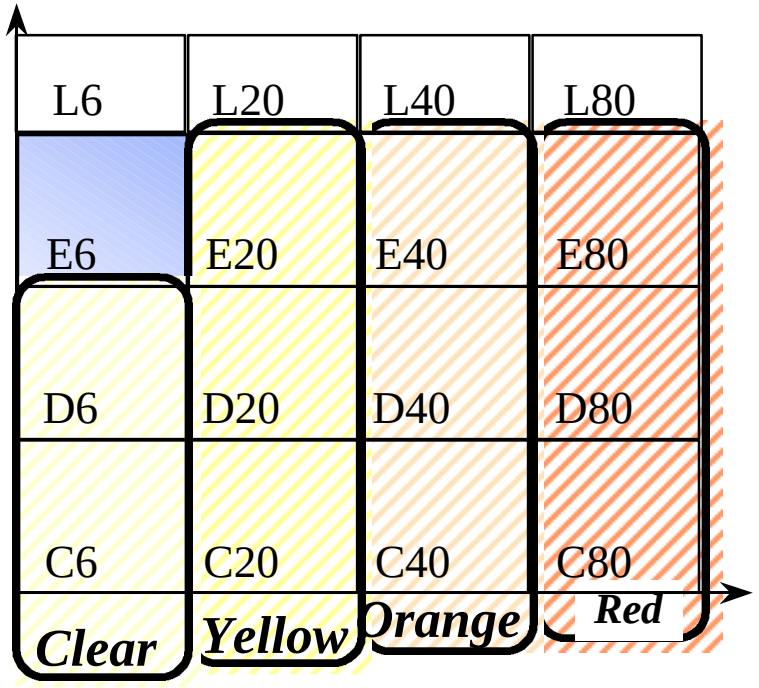
- Methodenfamilie von Alistair Cockburn
- Teamansatz
- Regelmäßige Reflexion und Anpassung
- Buchform geplant für 2002

Adaptive Software Development

- Managementansatz auf Basis „lernender Organisationen“
- Keine konkrete Methode
- Buchform: [Hig00]

Die Crystal Familie umfasst vier Methoden

- Einteilung nach Kritikalität und Teamgröße
- Leichteste Variante: Crystal Clear
- Brauchbar für:
  - Team < 8
  - Keine lebensnot-wendigen Systeme
  - Lokale Entwicklung



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

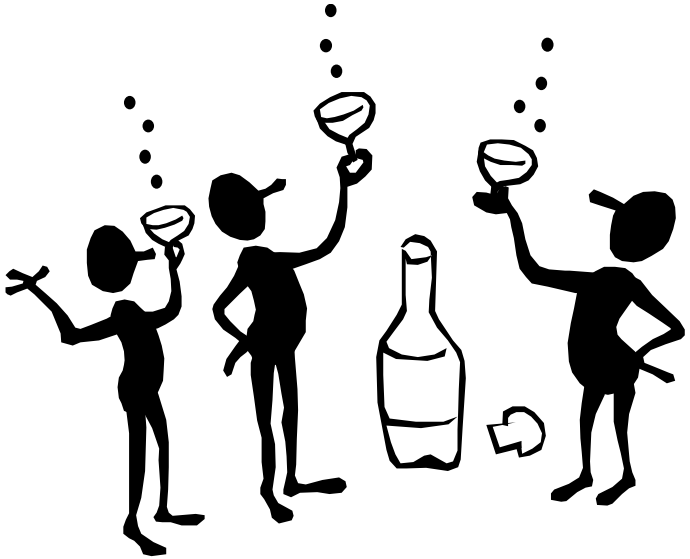
Crystal Clear hat das typische Auftreten agiler Prozesse

- Viel Kommunikation, wenig Artefakte
- Kommunikationswege werden kurz, effizient und informell gehalten
- Regelmäßige Auslieferungen (< 4 Monate)
- Ballast reduzieren
- Mehr Auslieferungen und bessere Kommunikation reduzieren die Notwendigkeit von Zwischenprodukten

Übersetzt nach einer Manuskriptversion von: Alistair Cockburn: *Crystal Clear,* [Coc02]

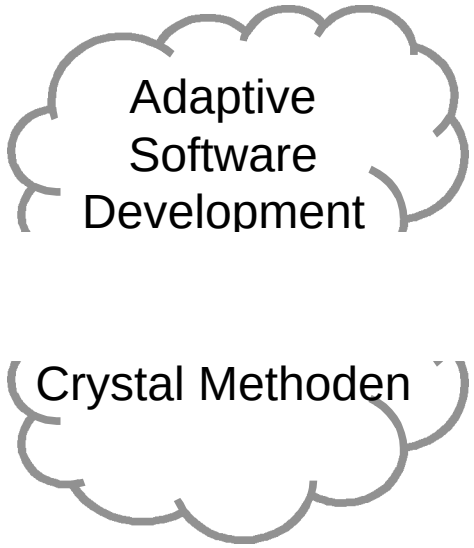
Besonderheit sind die expliziten Reflexionsrunden

- Mindestens zweimal pro Inkrement setzt sich das Team zusammen und reflektiert über das Vorgehen
- Dabei wird das Vorgehen den aktuellen Problemen und Erfahr-ungen angepasst.
- So entsteht eine „selbst-adaptierende“ Methode



Adaptive Software Development entspricht im wesentlichen Teil II

- Das Management stellt das Team auf, lässt es arbeiten und gibt ihm anschließend die Möglichkeit, zu lernen
- Basis: Peter Senges Ansatz „Lernender Organisationen“ [Sen90]
- Gute Ergänzung der Crystal Serie aus Managementsicht



Zusammenfassung

- Agile Prozesse bilden in vielen Bereichen eine Alternative zu klassischen Prozessen
- Sie erhöhen 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

Die Zukunft ist noch unklar

- „Methodenkrieg“ XP gegen Crystal/ASD?
- Oder einen neuen „Unified“ Prozess?
- In welchen Bereichen werden sich agile Prozesse durchsetzen?
- Wie endet der XP Hype?

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/

[Coc98] Alistair Cockburn: Crystal Clear; Addison-Wesley, Reading, Massachusetts, Veröffentlichung vorgesehen für 2002; Informationen und Vorabversionen unter http://www.crystalmethodologies.org

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

[GaW89] Donald C. Gause, Gerald M. Weinberg: Exploring Requirements - Quality Before Design; Dorset House, New York, 1989; ISBN 0-932633-13-7

[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

[Hig01] James A. Highsmith III: Highsmith And Cockburn; verfügbar auf: http://www.crystalmethodologies.org/cgi-bin/wiki.pl?HighsmithAndCockburn

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

[Wal90] Ernest Wallmüller: Software Qualitätssicherung in der Praxis; Carl Hanser Verlag München, Wien, 1990; ISBN 3-446-15846-4

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

Weitere Links zum Thema

- Dieser Vortrag: http://www.coldewey.com
- Agile Prozesse: http://www.agilealliance.org/
- Crystal/ASD: http://www.crystalmethodologies.org/
- eXtreme Programming: http://www.xprogramming.com/

