Prozeßmodelle

Vorlesung: Software-Engineering für große, betriebliche Informationssysteme

für Universität Leipzig, Sommersemester 2004

Institut für Software- und Systementwicklung
Professur Wirtschaftsinformatik, insbes. Informationsmanagement

Hans Hartmann (Generali VIS Informatik Ges.m.b.H., Wien)
Wolfgang Keller (AMB Generali Informatik Services GmbH, Aachen)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Überblick

- Einführung und Forces
- Wasserfallmodel
- Spiralmodell, inkrementelles Vorgehen
- Prototyping
- RUP - Rational Unified Process
- eXtreme Programming und andere agile Prozessmodelle
- Von der Theorie zur Praxis – der SEP (Software-Entwicklungsprozeß

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Warum brauche ich einen Softwareentwicklungsprozess

- Fall 1: Ich entwickle zu meinem persönlichen Spaß ein Schachprogramm als Freeware
- Fall 2: Eine große Firma entwickelt ein Schachprogramm, das den Weltmeister schlagen soll
- Fall 3: Leute die nichts von Schach verstehen entwickeln ein Schachprogramm, das zumindest ein Konkurrenzunternehmen schlagen soll

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 1: Persönlicher Softwareprozess

- Ich sitze allein vor meinem Rechner und möchte ein Freeware Schach-Programm für meinen Spaß entwickeln
- Fragen
  - Was tue ich da?
  - Brauche ich einen Entwicklungsprozess?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 1: Persönlicher Softwareprozess

- Diverse Fragen die ich mir selbst beantworten muß
  - für welche Zielumgebung schreibe ich mein Programm?
  - Wie soll es an der Oberfläche aussehen
  - Wie sieht meine grobe „Architektur“ aus
  - Welche Features möchte ich bieten
    - Das Programm soll gegen mich spielen können
    - es Partien analysieren könne
    - es soll auch Schachtrainer sein
  - Biete ich nur eine neue Oberfläche und nehme die GNU Engine oder schreibe ich eine eigene Engine?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 1: Persönlicher Softwareprozess

- Fragen, die ich mir oft nicht beantworte
  - wie lange werde ich brauchen?
  - was wird es kosten
  - welche Spielstärke wird das Programm in Elo-Punkten haben
  - Was passiert, wenn es die nicht hat
  - soll ich eine formale Spezifikation verwenden oder nicht

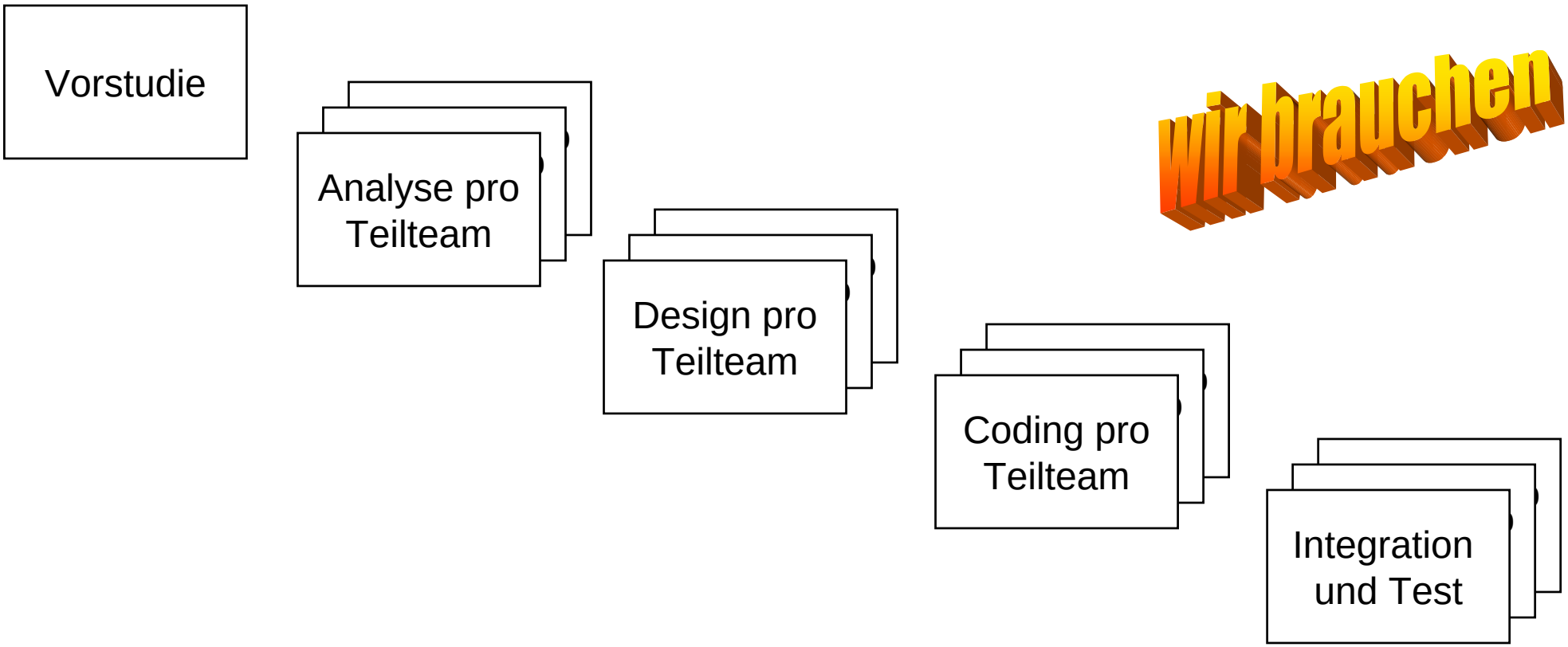
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Die große Firma möchte den Weltmeister schlagen

- Es wird zunächst eine Machbarkeitsstudie erstellt werden, anhand derer das Problem vorstrukturiert wird
- Dann wird eine grobe Architektur erstellt, anhand derer auch die Teilteams aufgeteilt werden
- Jedes Teilteam analysiert seinen Bereich, entwirft ein Design dafür und implementiert und testet die Umsetzung
- am Ende passt hoffentlich alles zusammen

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

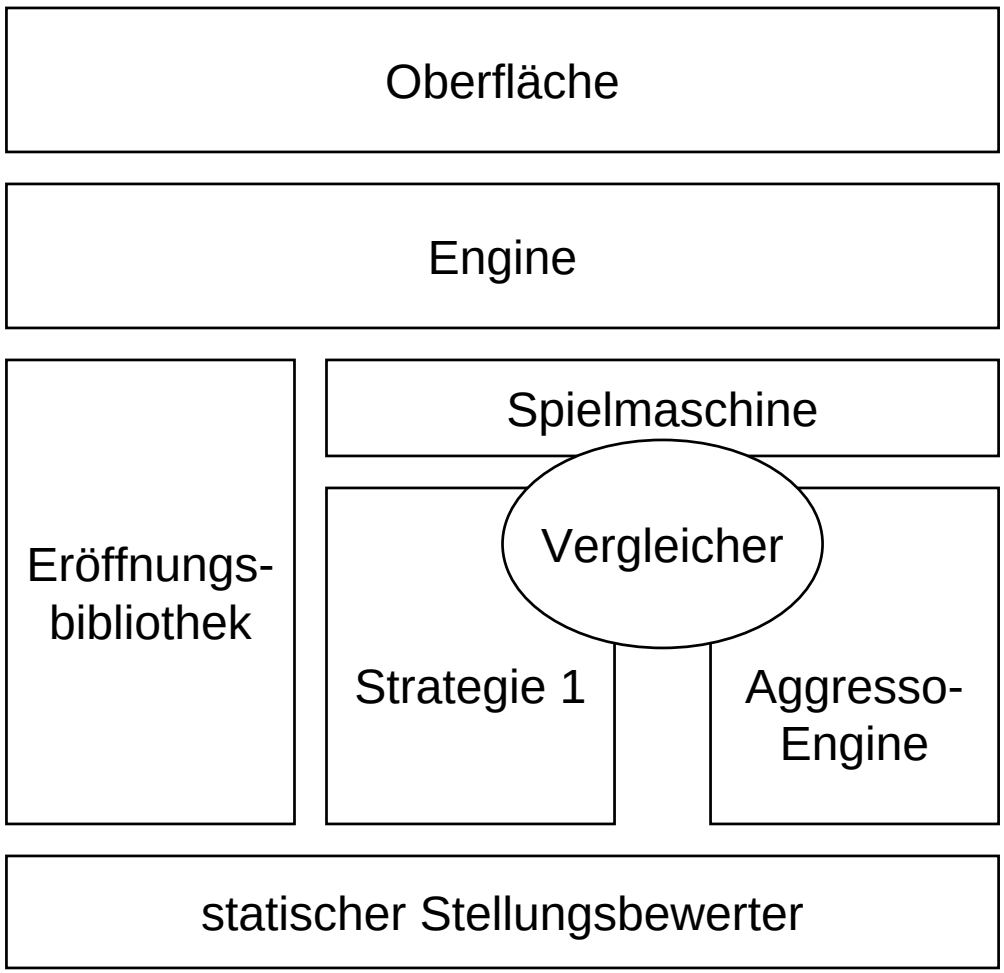
Fall 2: Deep Chess Architektur (fiktiv)



und es geht noch weiter

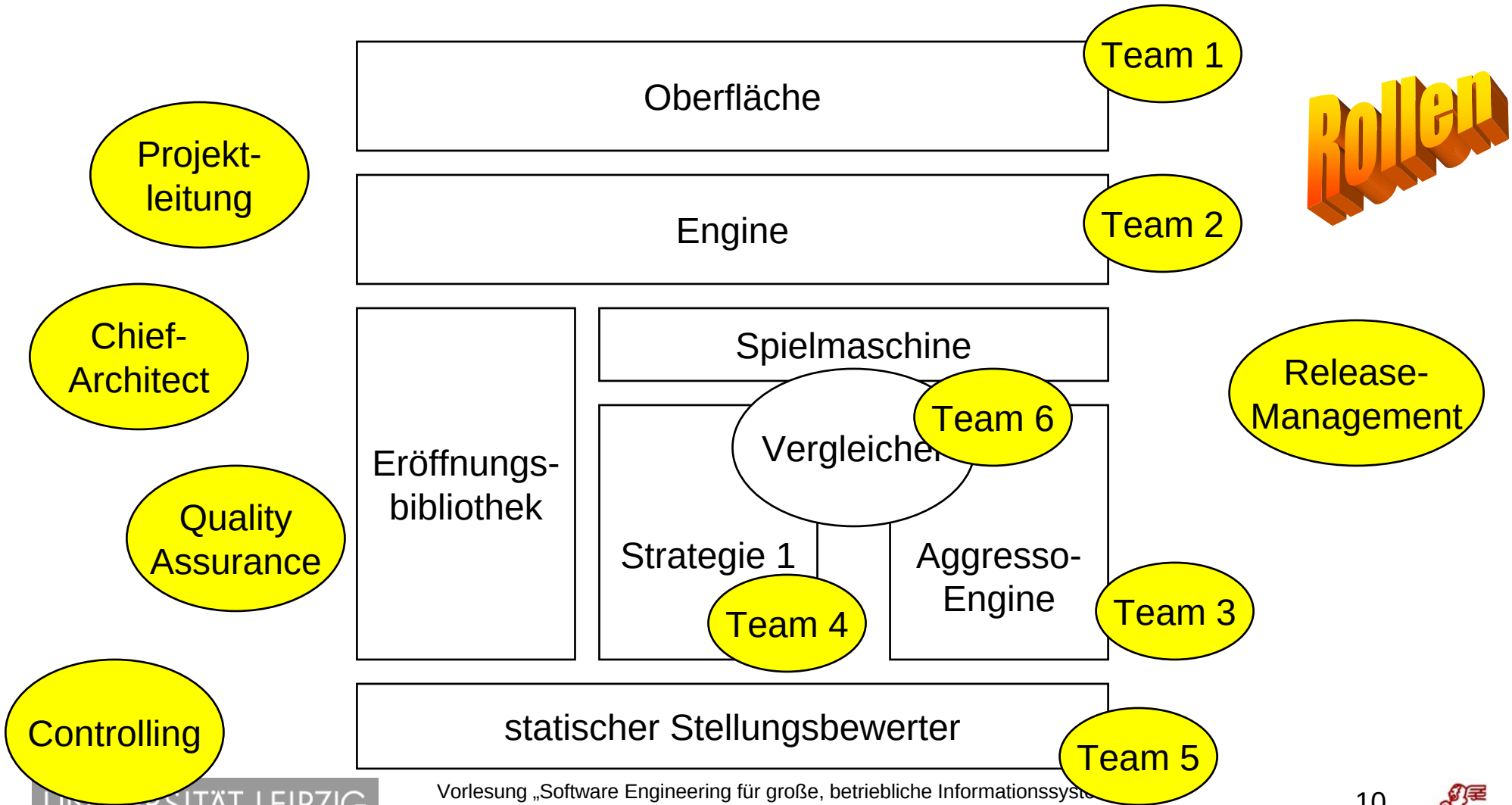
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Deep Chess Architektur (fiktiv)



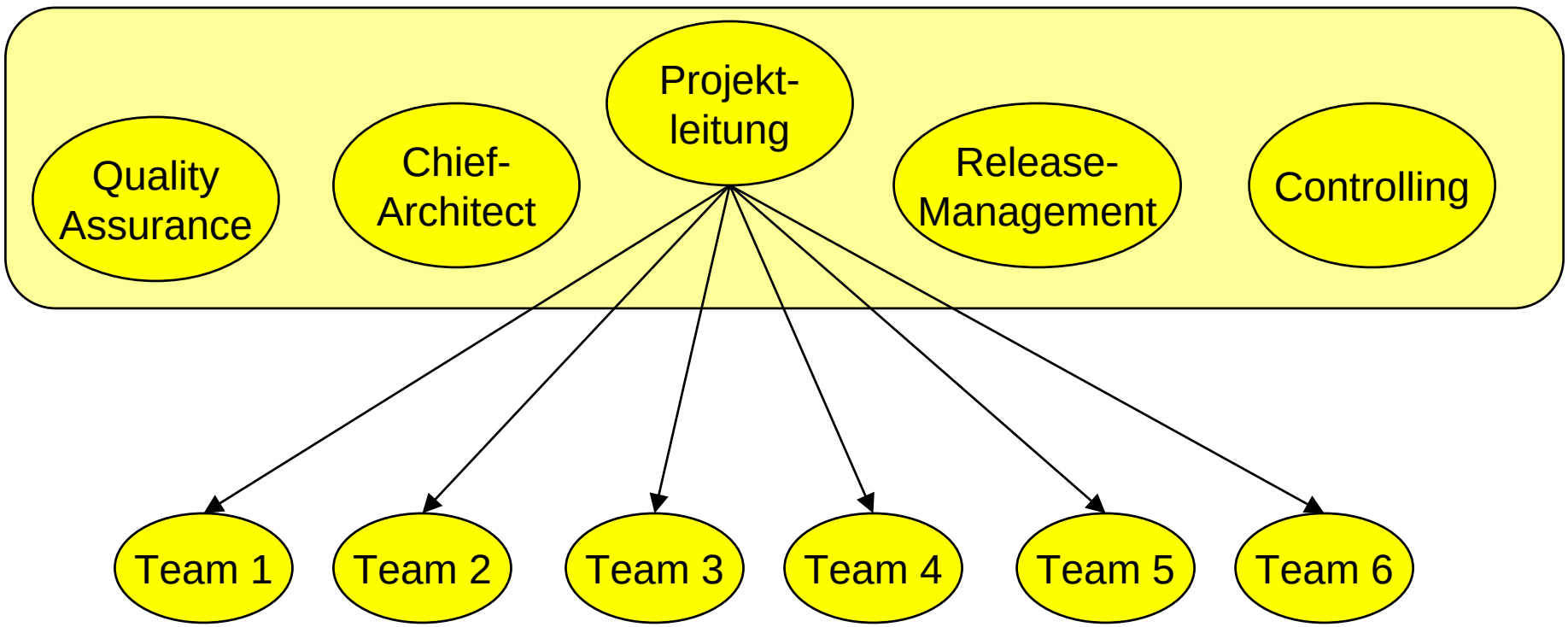
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Deep Chess Architektur (fiktiv)



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Deep Chess Architektur (fiktiv)

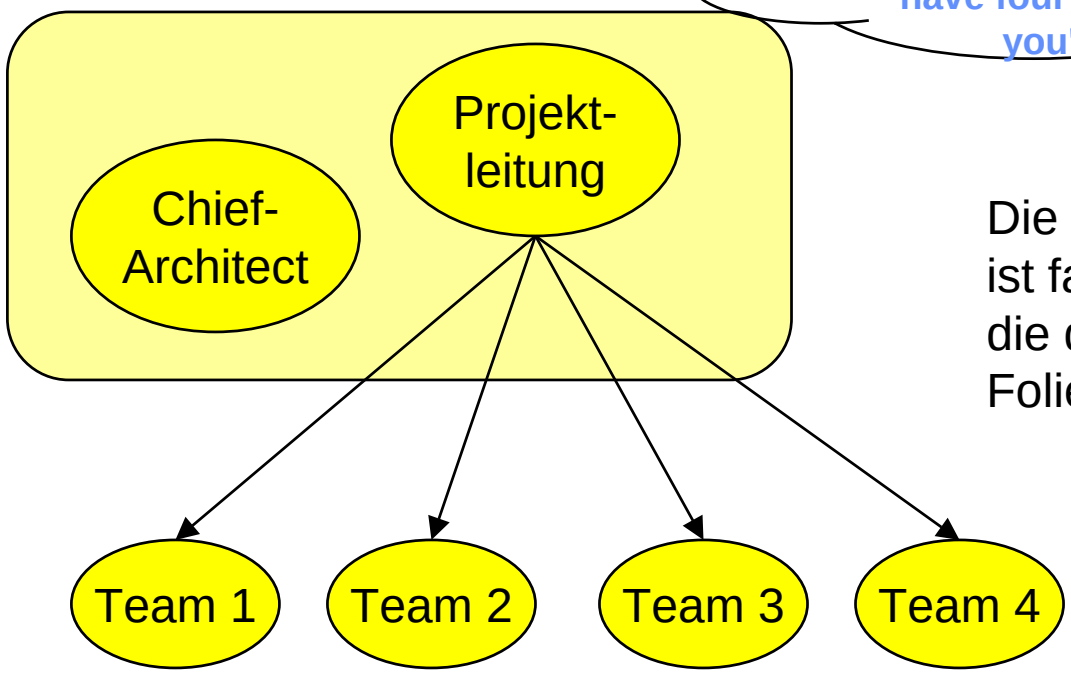


© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Deep Chess - Exkurs

Conways Law: Architecture follows Organization

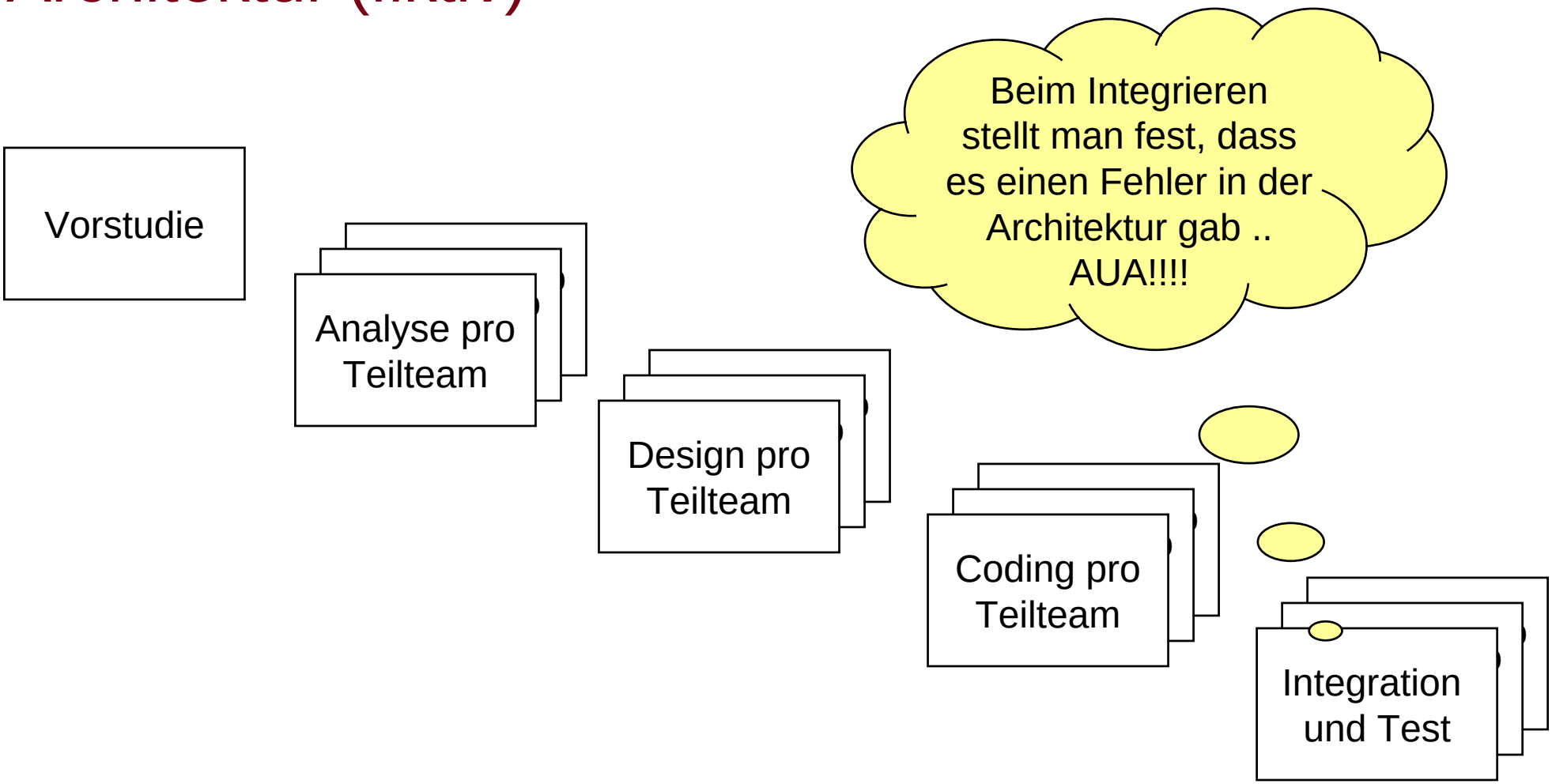
The rule that the organisation of the software and the organisation of the software team will be congruent; originally stated as "If you have four groups working on a compiler, you'll get a 4-pass compiler".



Die Architektur dieses Teams ist fast 100% sicher eine andere als die des Projektes auf der vorigen Folie

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 2: Deep Chess Architektur (fiktiv)



und es geht noch weiter

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein paar Einflußfaktoren also ..

- Der optimale Prozess hat etwas damit zu tun, wieviele Leute entwickeln
- Der Prozess ist damit beeinflusst von der Teamgröße
- Die Teamgröße wird von den Requirements und der Deadline beeinflußt
- Die Teamgröße wird von der Projektgröße beeinflusst (in Function Points)
  - $MM = f_{Language}(FP)$, $TG_{opt} = SQRT(MM)$, 1 MM = 1 PersonenmonatIn

- Die Architektur ist auch Folge eines sozialen Prozesses

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fall 3: 10 Leute, die von Schach nichts verstehen, bauen ein Schachprogramm ...

- Sie werden die Fachabteilung fragen und mit der eine Vorstudie und Analyse machen
- Alles wird länger dauern, als in Fall 2
- Es wird mehr Fehler geben
- Das Ergebnis wird schlechter ..

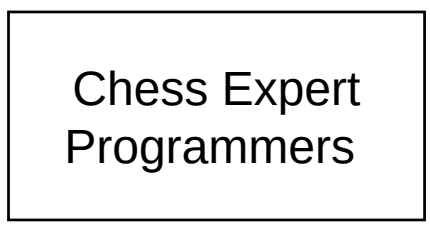
  … und eigentlich wird das Ergebnis nicht nur schlechter sein sondern schlichtweg ein Mist, was im vorliegenden Fall zumindest einigen Leuten die Gelegenheit geben wird, das Schachprogramm noch zu schlagen, was ja bei Fritz schon lange nicht mehr der Fall ist.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Weitere Einflussgrößen also

- Das Wissen der Beteiligten über das zu produzierende Objekt beeinflusst die Durchlaufzeit und den Prozess

Wer ist wohl schneller????



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Frage dann noch ..

- Wie viel Sinn macht ein absolut einheitlicher Prozess für jede Art von Projekt in einer Firma?
- Es gibt mehr als eine Firma, die diesem Ideal hinterherläuft?
- (sinnvoller) Begriff dazu: Tayloring. Der Prozess wird an das Projekt angepasst, bevor es losgeht ...

Das kriegen wir dann später noch einmal ,

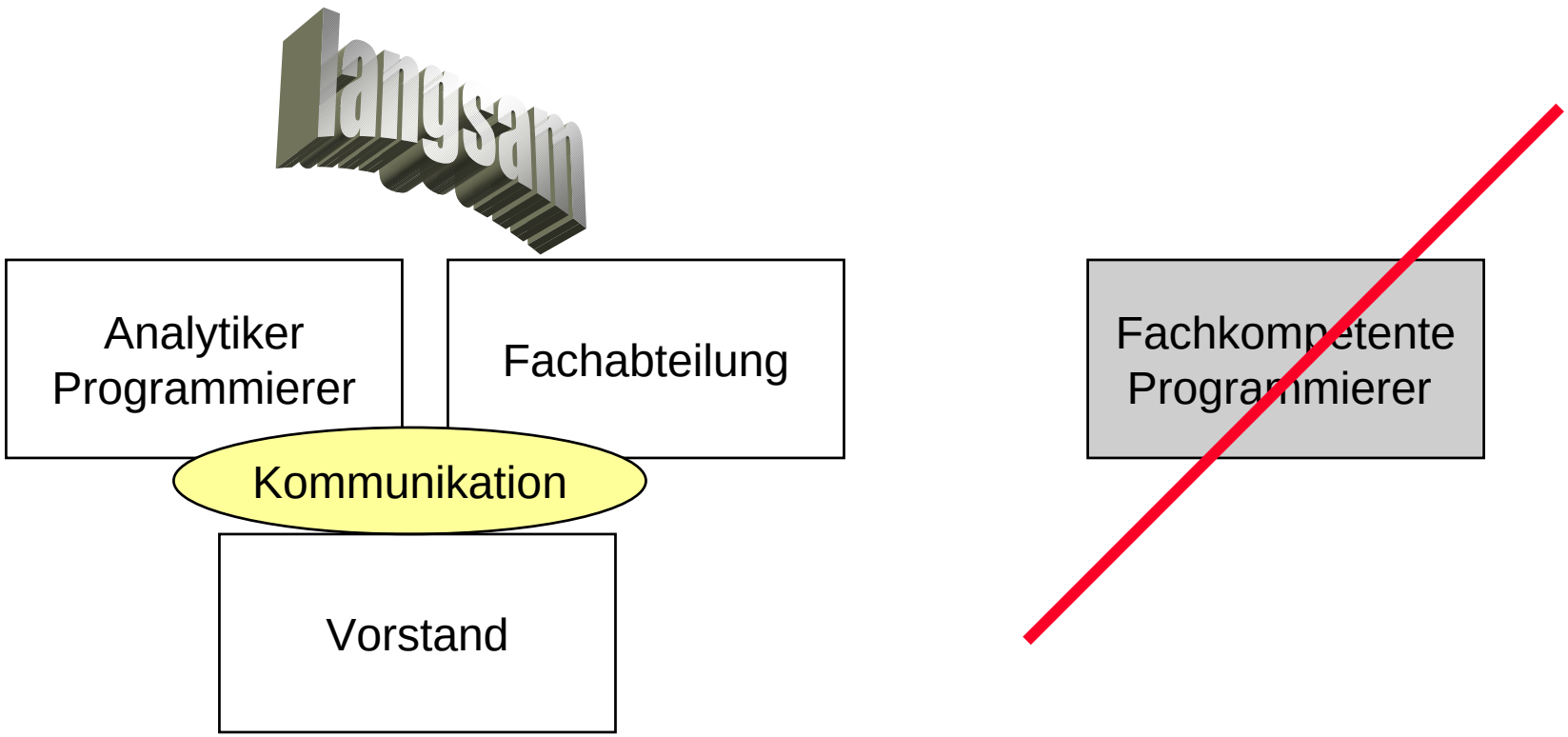
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ideal...

Fachkompetenter
Programmierer
als Anwender

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

... Realität



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

„Separation of Concerns“ Rollen

Sie brauchen....	Rollenname	... Aus der Linie im Orgchart
•Jemand, der einen Nutzen erwartet und Ihr Projekt finanziert	Sponsor Auftraggeber	Vorstand
•Jemand, der plant, nachläuft, mahnt, kontrolliert, dafür sorgt, daß was geschieht	Projektleiter	Fachbereich EDV (beide)
•Jemand, der die fachliche Lösung konzipiert	Analytiker Key-User	Fachbereich EDV-Beratung
•Jemand, der die technische Lösung konzipiert	Architekt	EDV
•Jemand, der die Lösung programmiert	Entwickler	EDV
•Jemand, der die Lösung testet	Tester	Fachbereich
•...		

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Das Problem: Worüber reden?

Sie brauchen....	Rollenname	... Aus der Linie im Orgchart
•Jemand, der einen Nutzen erwartet und Ihr Projekt finanziert	Sponsor Auftraggeber	Vorstand
•Jemand, der plant, nachläuft, mahnt, kontrolliert, dafür sorgt, daß was geschieht	Projektleiter	Fachbereich EDV (beide)
•Jemand, der die fachliche Lösung konzipiert	Analytiker Key-User	Fachbereich EDV-Beratung
•Jemand, der die technische Lösung konzipiert	Architekt	EDV
•Jemand, der die Lösung programmiert	Entwickler	EDV
•Jemand, der die Lösung testet	Tester	Fachbereich
•...		

Unterschiedliche Kulturen
=
Kommunikationsprobleme

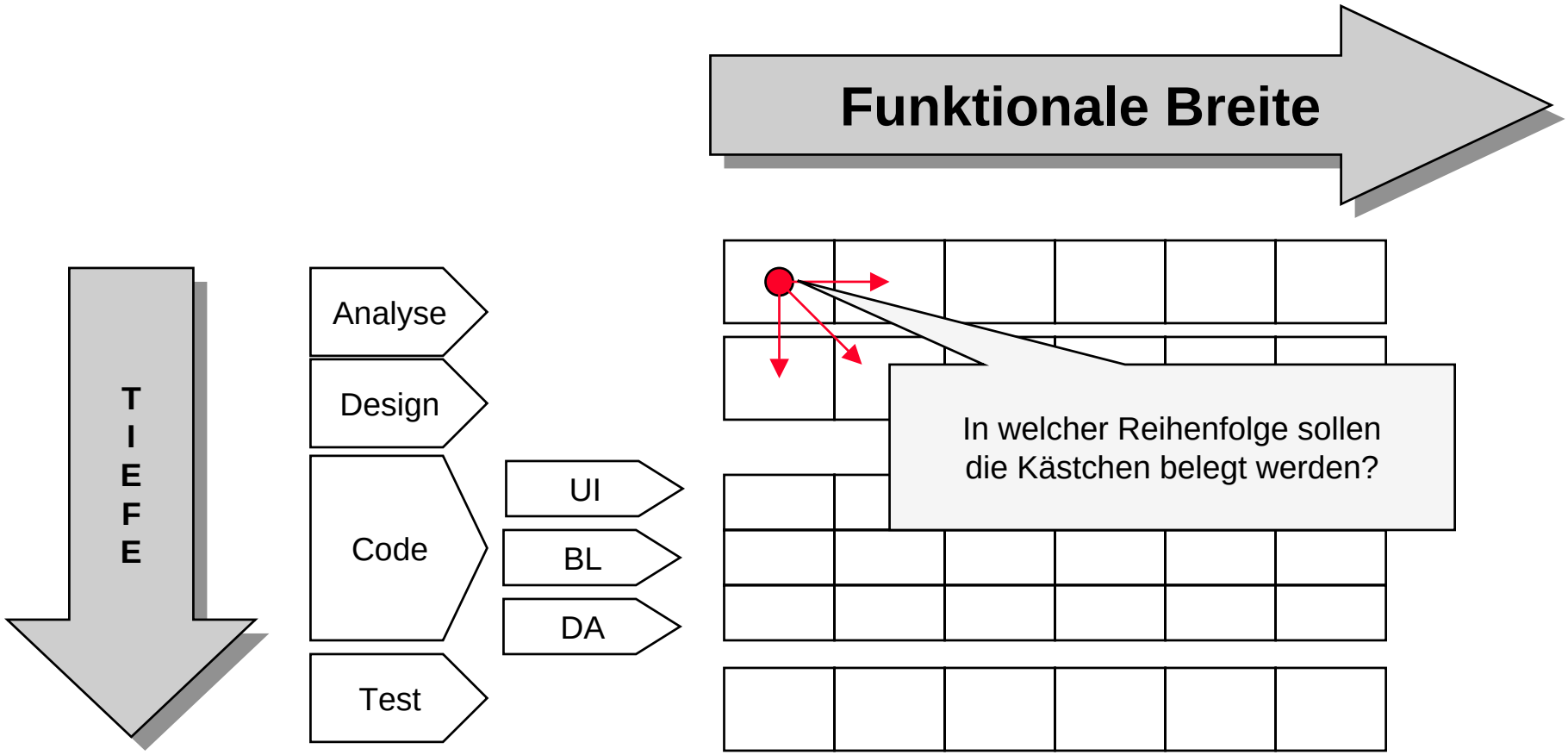
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prozesse Organisieren die Kommunikation anhand geregelter Flüsse von Teilergebnissen

Teilergebnis	Rollenname	... Aus der Linie im Orgchart
•Business Case	Sponsor Auftraggeber	Vorstand
•Projektplan	Projektleiter	Fachbereich EDV (beide)
•Use Cases •... (siehe Spezifikation)	Analytiker Key-User	Fachbereich EDV-Beratung
•Architektur, Design	Architekt	EDV
•Code	Entwickler	EDV
•Testfälle	Tester	Fachbereich
•...		

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prozesse beschreiben die Reihenfolge der Entwicklung



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Weitere Begriffe im Umfeld Entwicklungsprozeß

	Löst das Problem	Besteht meist aus	Scope
Methode	•Welche Ergebnisse sollen produziert und abgestimmt werden	**•Fluß von definierten Zwischenergebnissen**	**Allgemein**
Prozeß	•Welche Rolle wirkt wie bei der Erstellung welches Ergebnisses	**• Phasen** **• Iterationen** **• Rollen**	**Projekt Spezifisch**
Plan	•Wer macht was wann	• Ressource-Allocationen • Meilensteinen • Tasks	

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein persönlicher Tipp...

Und nicht auf die Kontrolle der Ergebnisse vergessen!

Löst das Problem

Besteht meist aus

Scope

Methode

Prozeß

Plan

„Planen Sie die Ergebnisse, nicht die Arbeit“.

Allgemein

Projekt Spezifisch

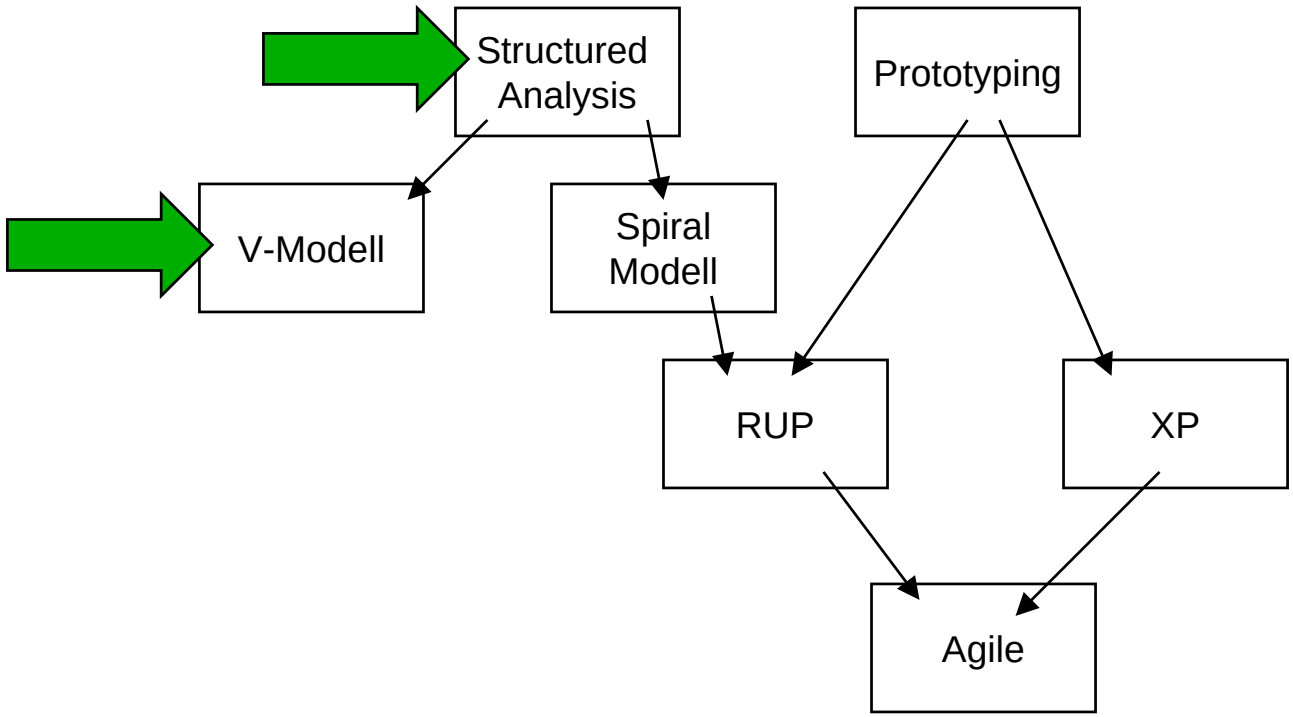
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Einflußfaktoren (– eine Wiederholung)

- Der optimale Prozess hat etwas damit zu tun, wieviele Leute entwickeln und miteinander kommunizieren.
- Der Prozess ist damit beeinflusst von der Teamgröße.
- Die Teamgröße wird von der Projektgröße und der Deadline beeinflußt.
- Aber! Die Effizienz der Kommunikation hängt von der Teamgröße ab.
- Die Projektgröße ist hingegen von der Architektur beeinflusst
- Gibt es da eine Schleife?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse

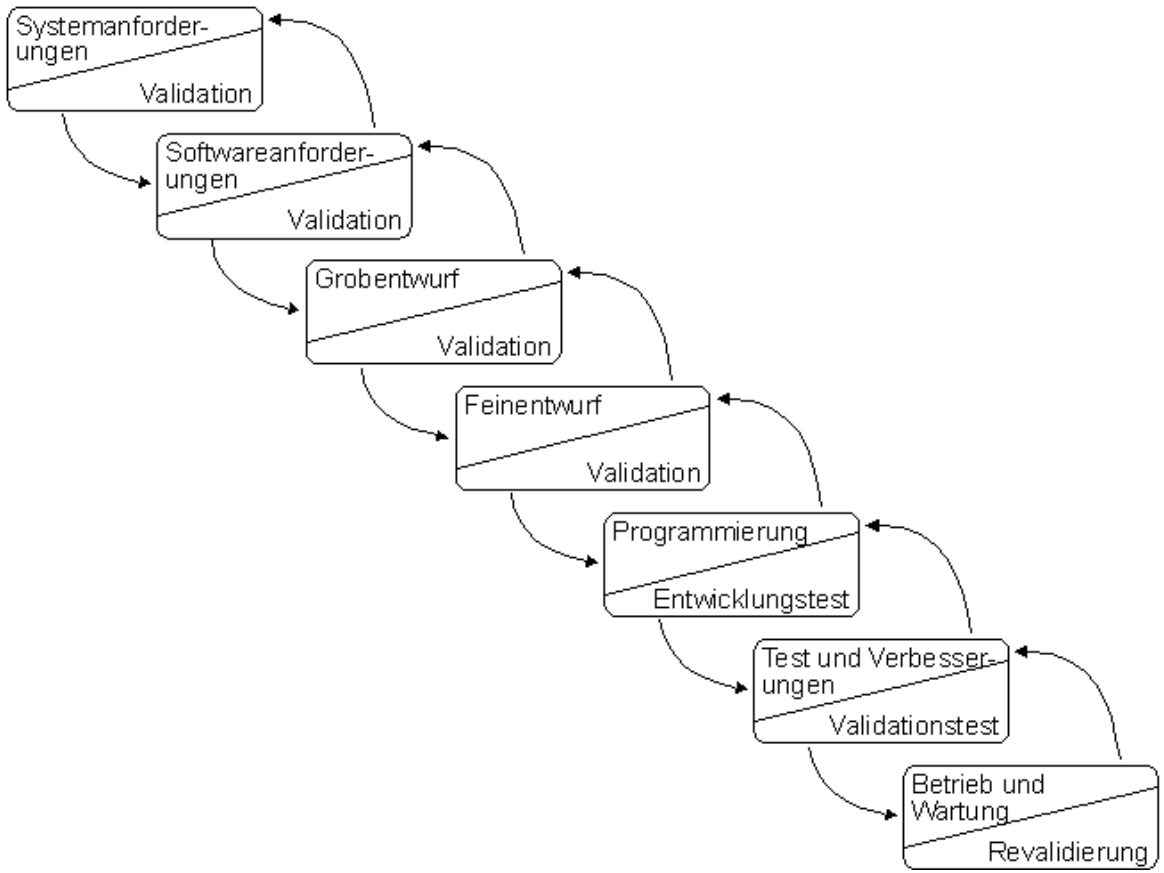


© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfallmodell:
Structured Analysis, V-Modell

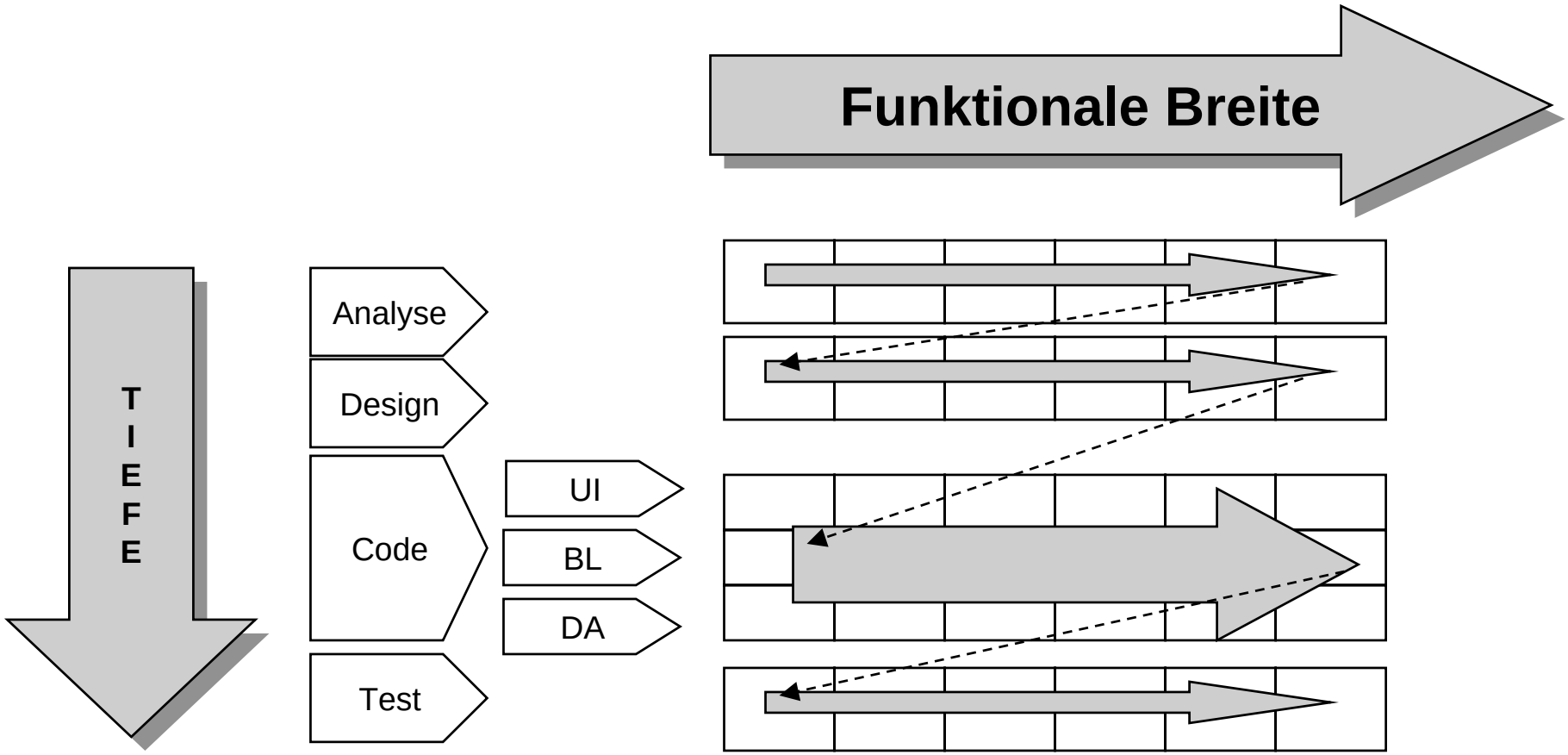
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall



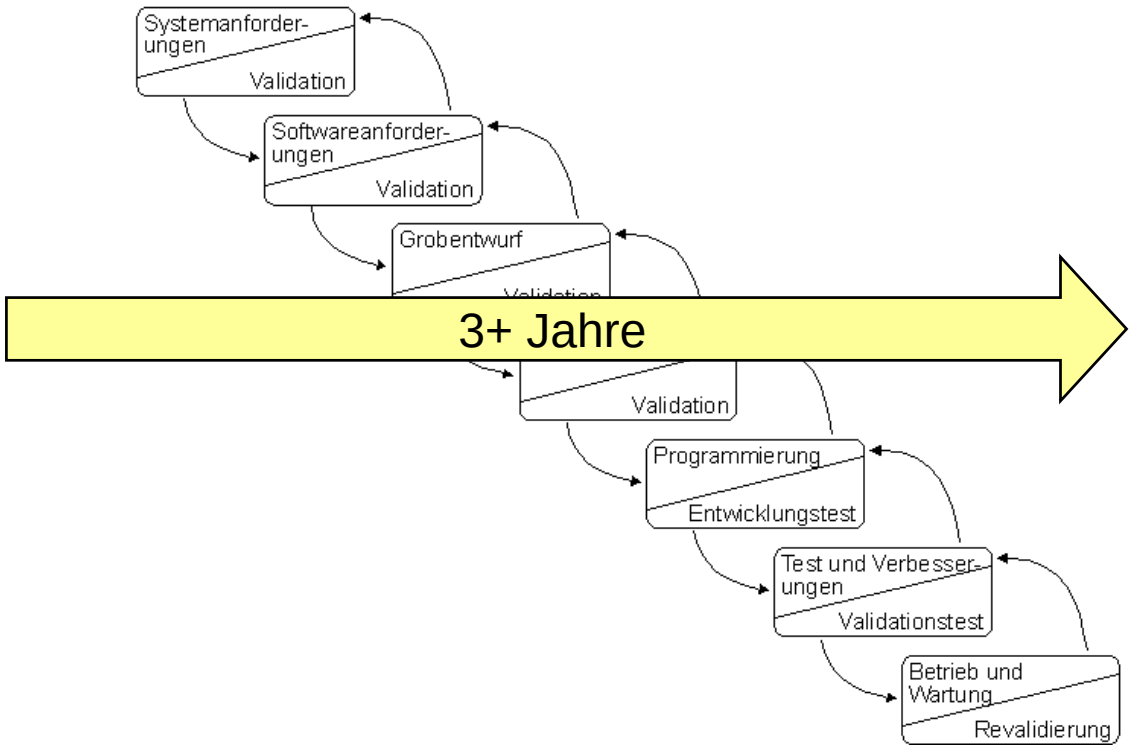
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall - Kästchenbelegunsstrategie



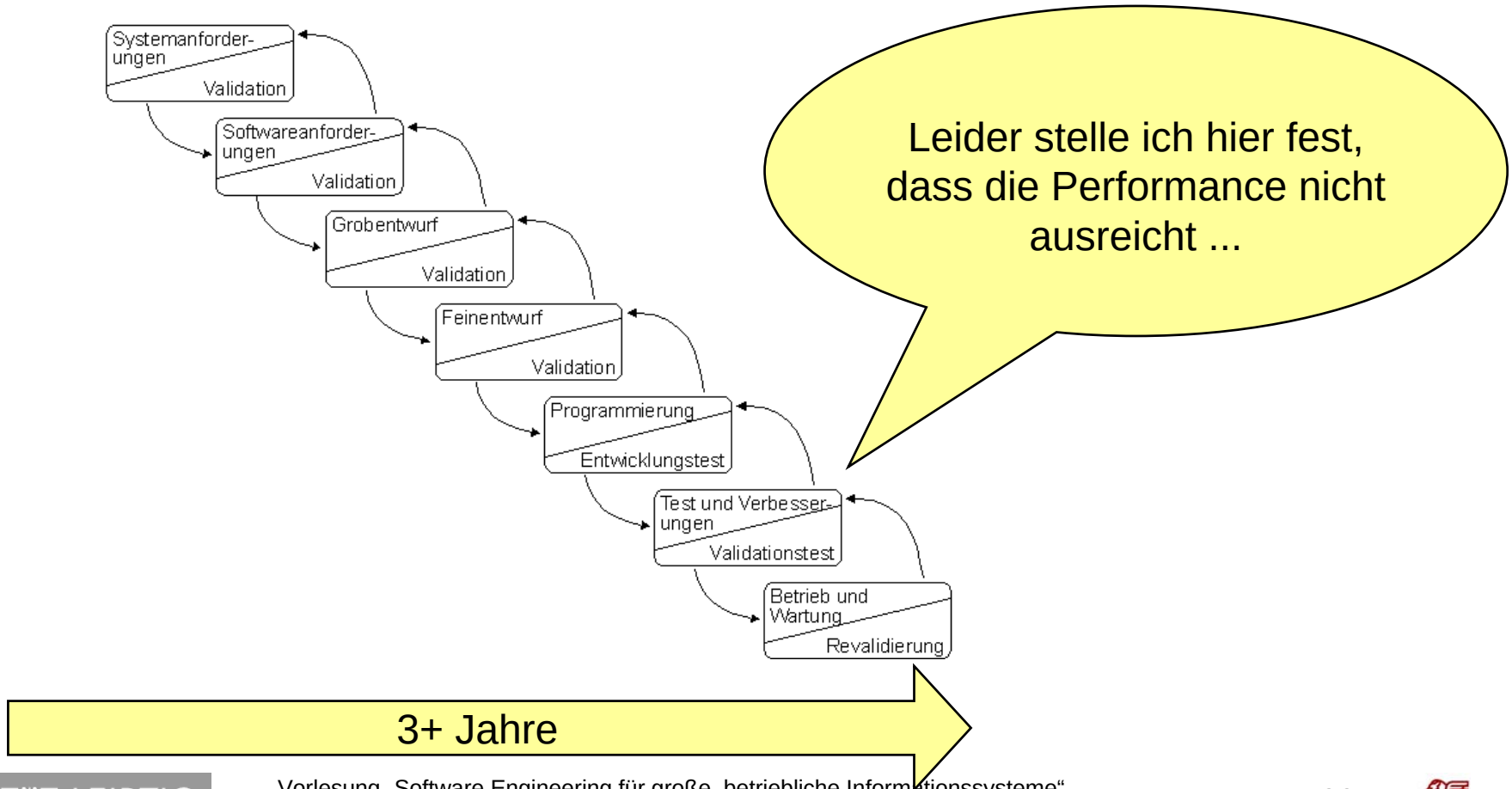
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall - eine schlechte Eigenschaft



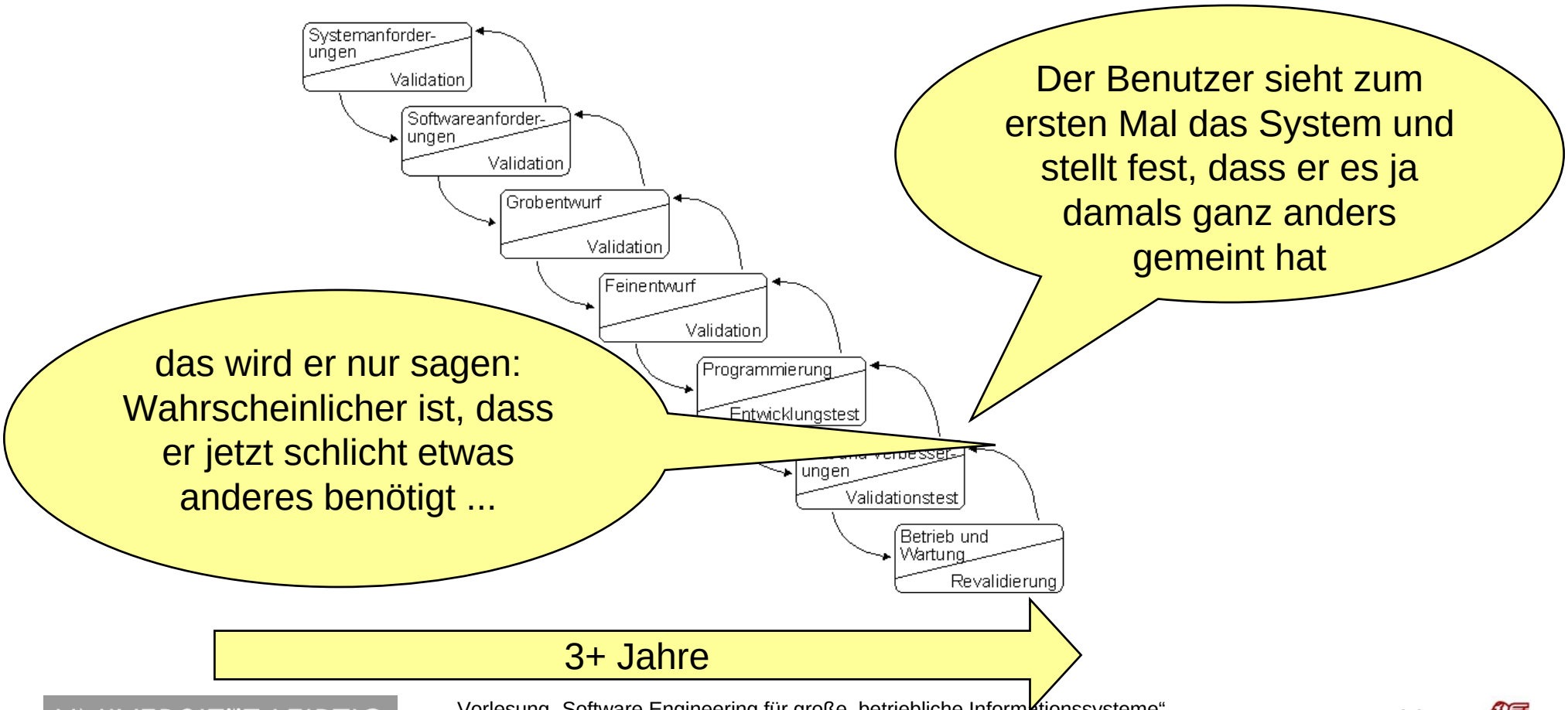
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

... Mit Konsequenzen ...



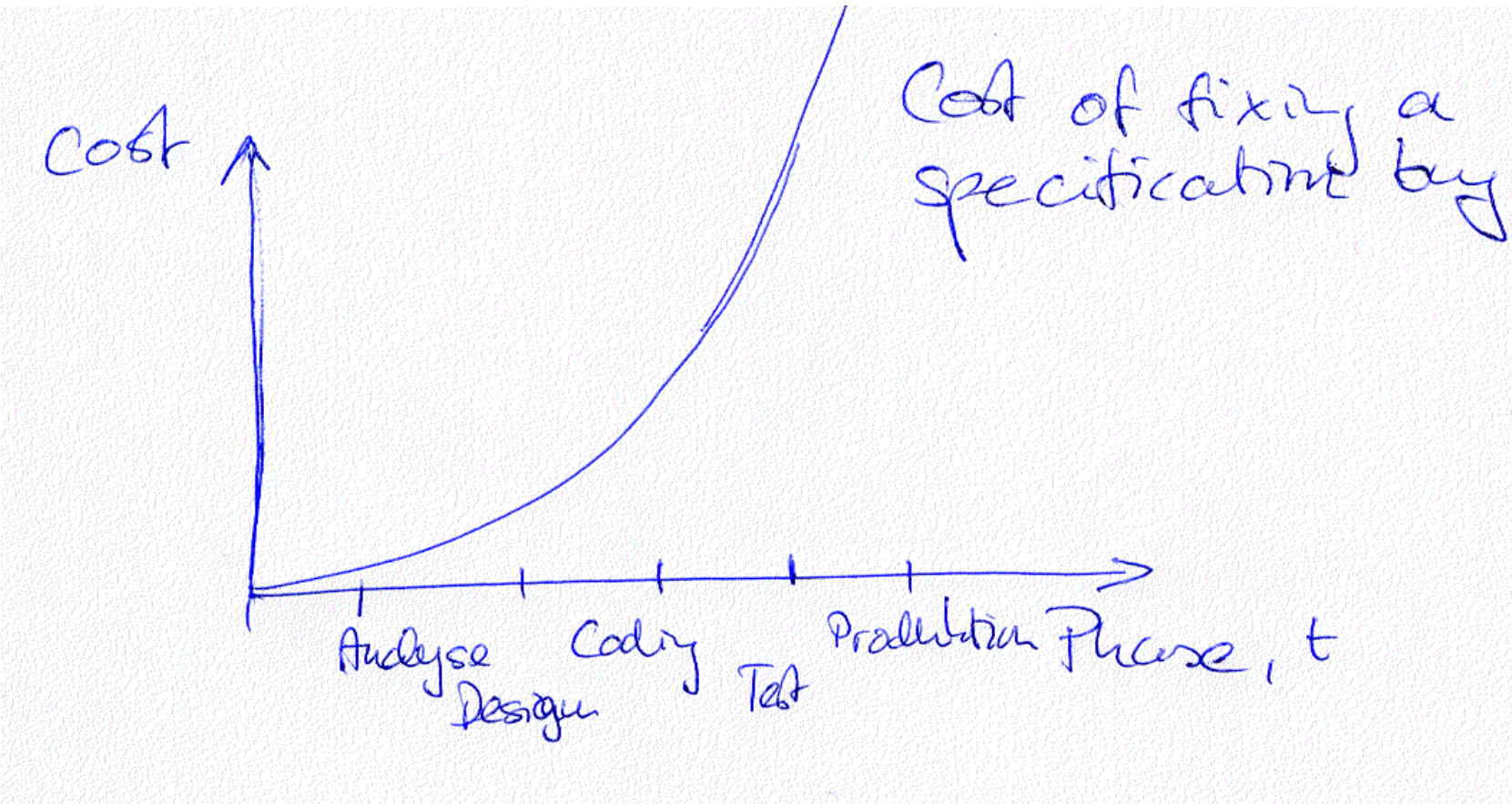
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

... Mit bitteren Konsequenzen ...



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Frühere Annahme (70er) .. Fehlerkosten steigen exponentiell



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall – Diskussion

Was zum Beispiel in der Generali in 3 Jahren passiert ist

- Beispiel: Das Lebenssystem für die Gruppe Wien
  - 1999: Generali München geht an einen anderen Teilkonzern Unternehmen (Aachen Münchener)
  - 2000: Neue Vertriebswege führen zu neuen Produkten (Clever Invest)
  - 2001: Börsencrash führt zu Rückbesinnung auf Garantieprodukte
  - und so weiter und so weiter

- Wie wahrscheinlich ist es also, dass heute das benötigt wird, was ich vor 3 Jahren spezifiziert habe => NULL

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall - Vorteile

- die Tätigkeiten der Systementwicklung werden in einer sinnvollen Reihenfolge angeordnet
- Die Tätigkeiten und Dokumente können standardisiert werden. Damit können große Teams arbeiten. Das ganze ist einfacher zu managen
- Phasenübergang erfolgt nach Validierung und Freigabe (Qualitätssicherung)
- Damit hat man Milestones und eine Fortschrittskontrolle – einfach zu managen

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wasserfall - Nachteile

- Der Kunde sieht das Produkt sehr spät – bis dahin muss er es sich vorstellen – das überfordert 99%.
- Die meisten Kunden wollen keine Spezifikationen lesen – sie wollen Software.
- Linearer Durchlauf der Phasen ohne Änderungen ist sehr sehr unwahrscheinlich, weil sich wahrscheinlich die Vertragsgrundlage ändern wird
- Zusammen mit „alten“ Programmierumgebungen hat man das Problem der exponentiellen Änderungskosten

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

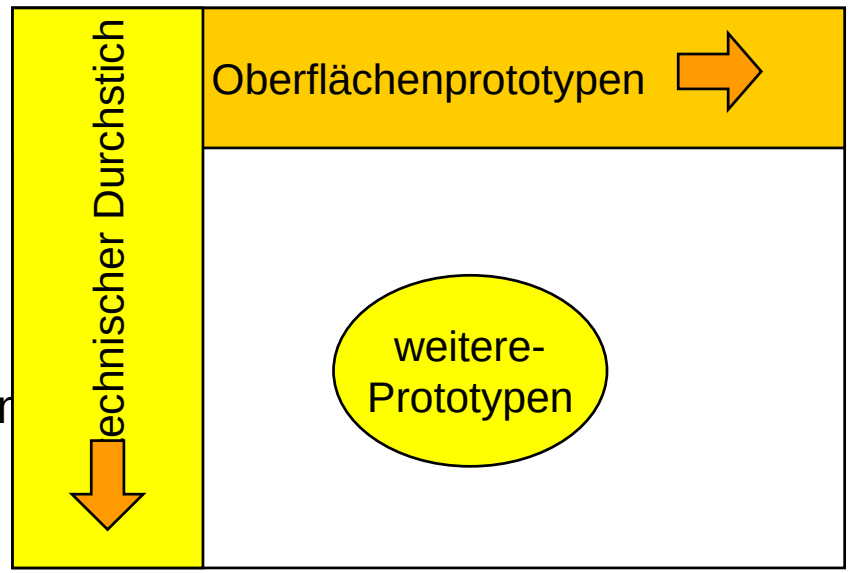
Wasserfall - Nachteile

- Die Dokumente der einzelnen Phasen sind redundant. Änderungen der Geschäftsgrundlage werden über die Zeit nicht nachgezogen. Damit degeneriert die Dokumentation

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

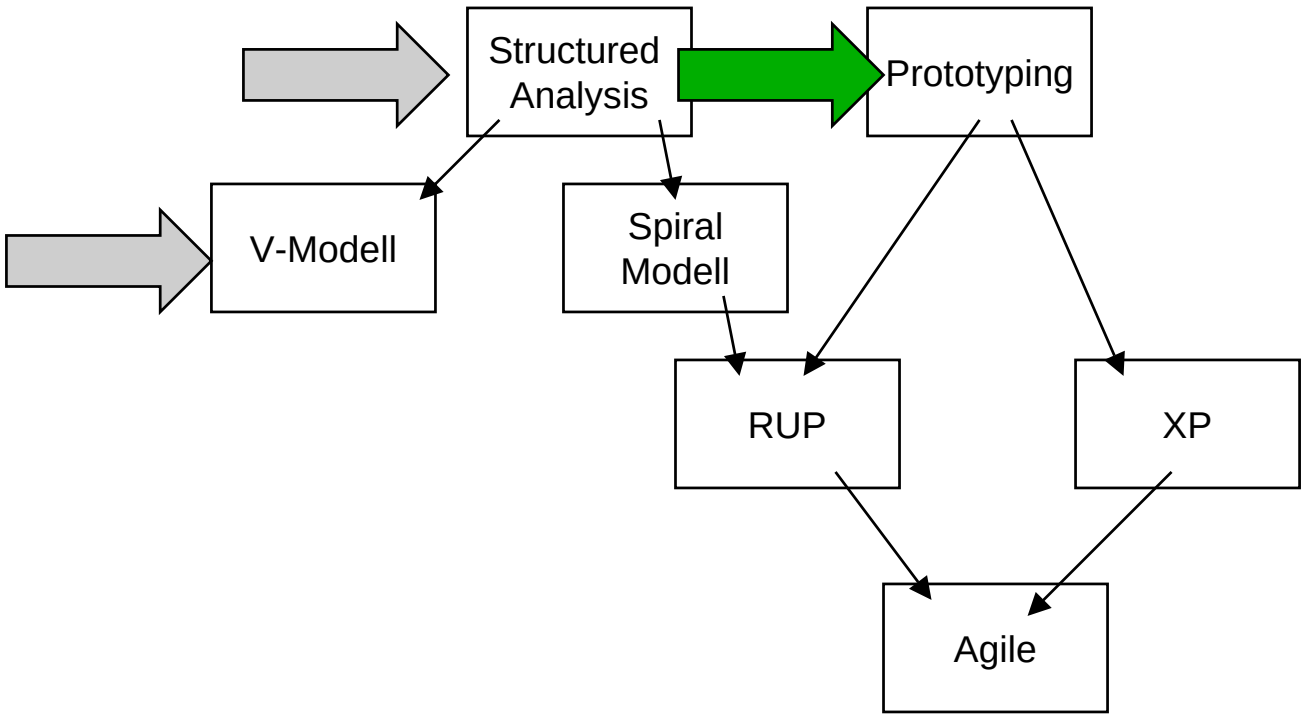
Idee: Prototyping

- durch technische (vertikale) Prototypen kann man vermeiden, dass man zum Beispiel die Performance Überraschung zu spät erlebt
- durch horizontale Prototypen sieht der Kunde früher was man bekommt
- komplexe Stellen sollte man mit weiteren Prototypen früh angehen („hardest things first“) – man merkt dann früh, welche Probleme man angehen muss



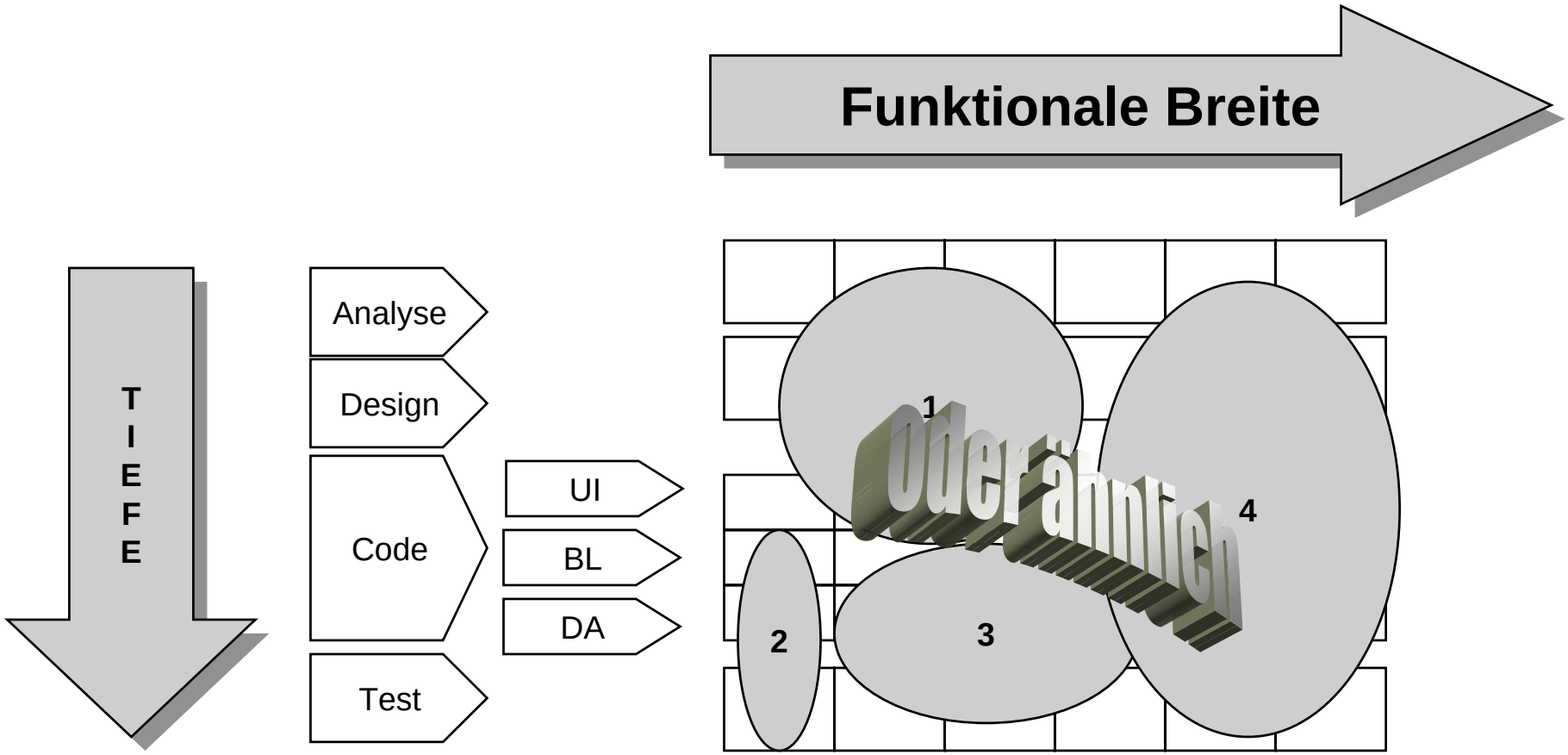
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prototyping - Kästchenbelegungsstrategie



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prototyping ...
Vorteile

- man kann damit Risiken minimieren und frühzeitig Feedback vom Endbenutzer bekommen
- Prototypen lassen sich gut in Prozessmodelle integrieren
- Oberflächen Prototypen lassen sich mit RAD Werkzeugen schnell bauen
- Sie liefern wichtige Information über das Gesamtsystem (auch Grundlagen für FP Schätzungen)

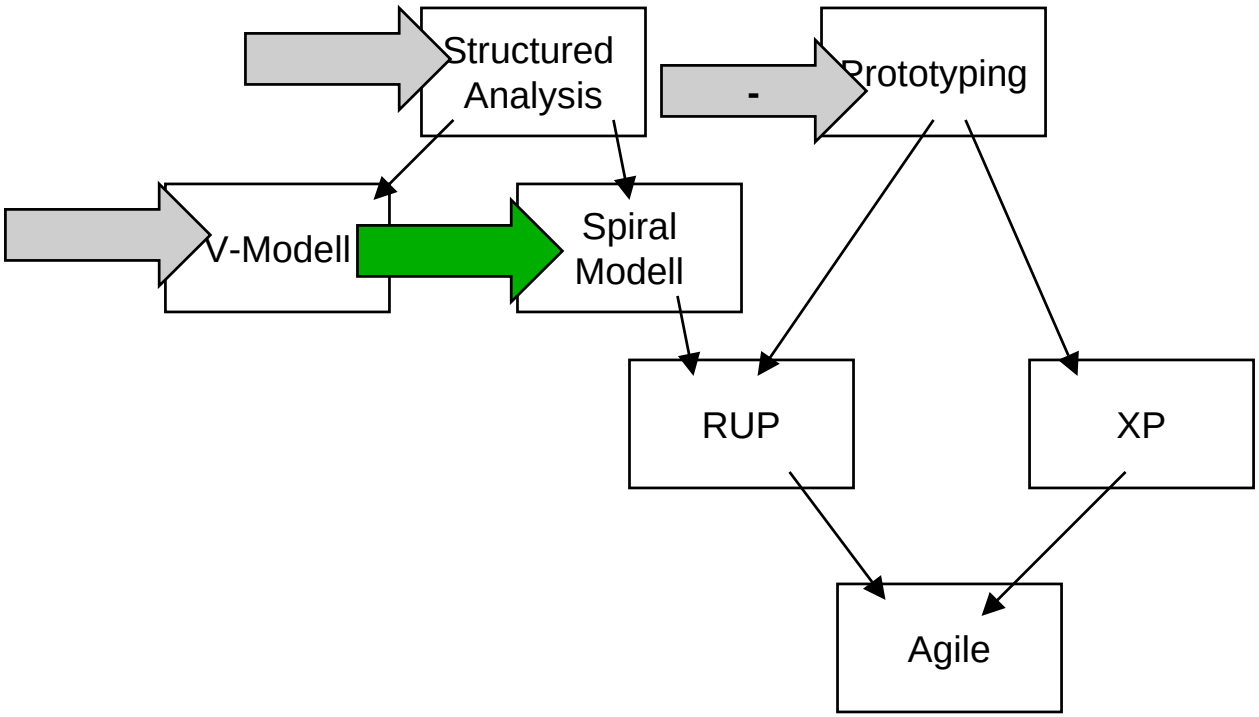
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Prototyping ...
Nachteile

- Der Entwicklungsaufwand steigt, da von den Prototypen viel weggeworfen wird
- Es soll schon vorgekommen sein, dass das Management einen Wegwerfprototypen als das fertige Produkt gesehen hat ☺
- und damit läuft man Gefahr, beim Management unter Druck zu geraten ..

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse

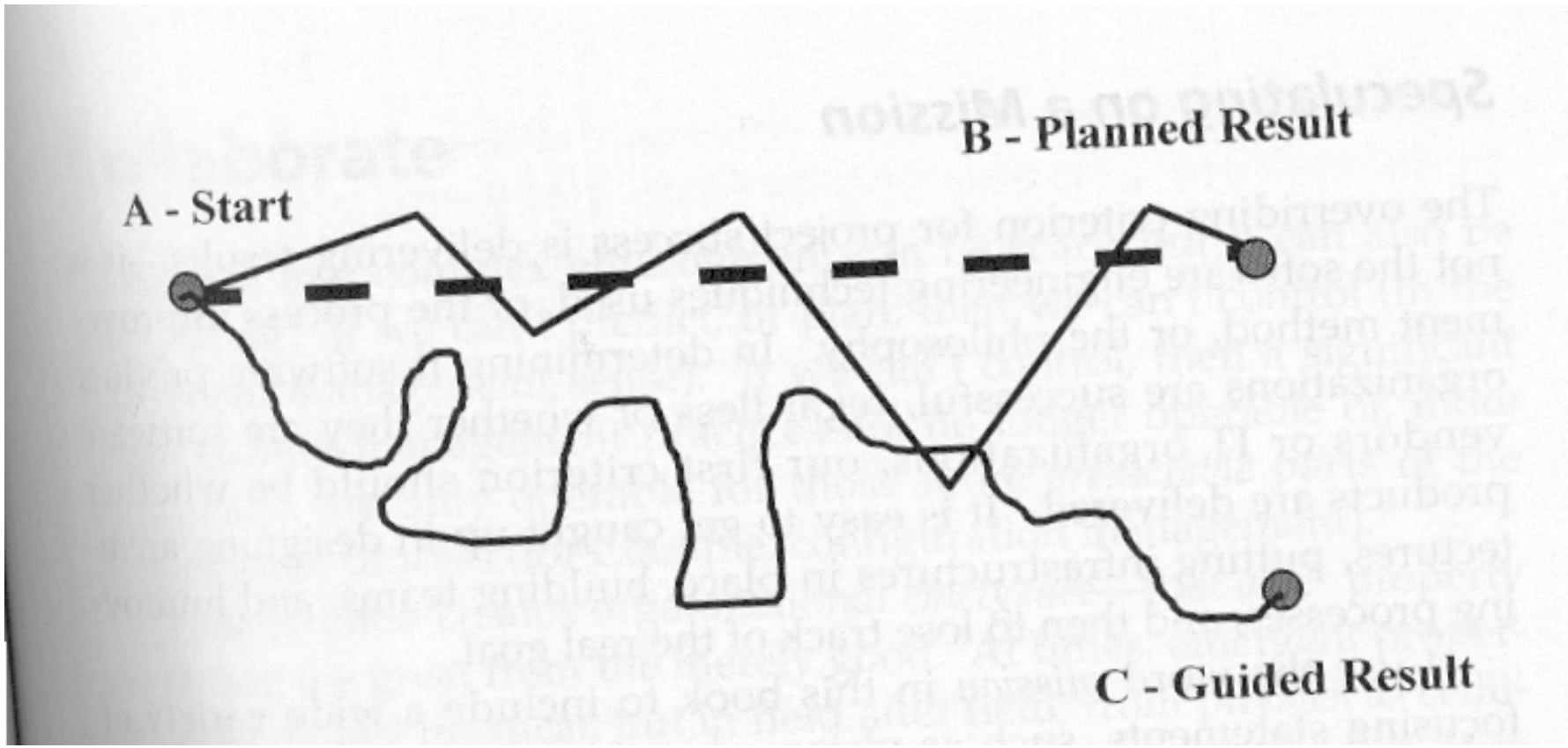


© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

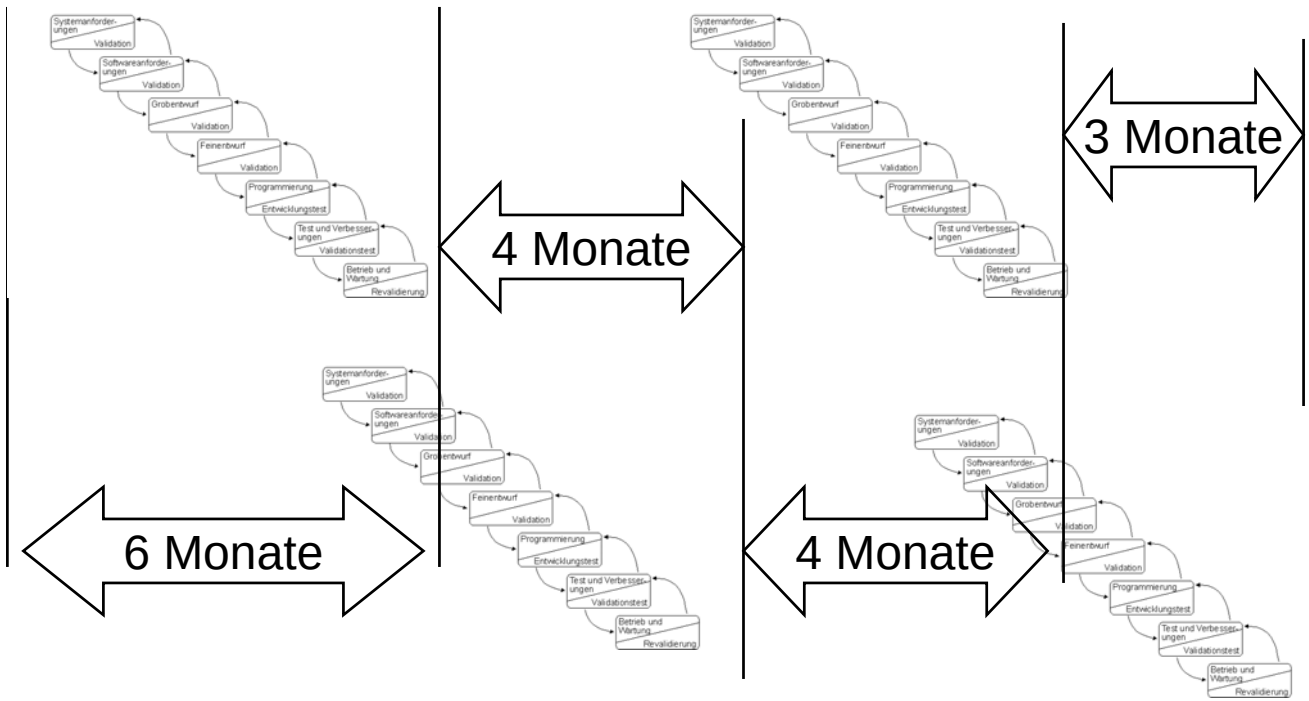
wesentliches Problem war ja:
Das Ziel ändert sich ...



[High2000]

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

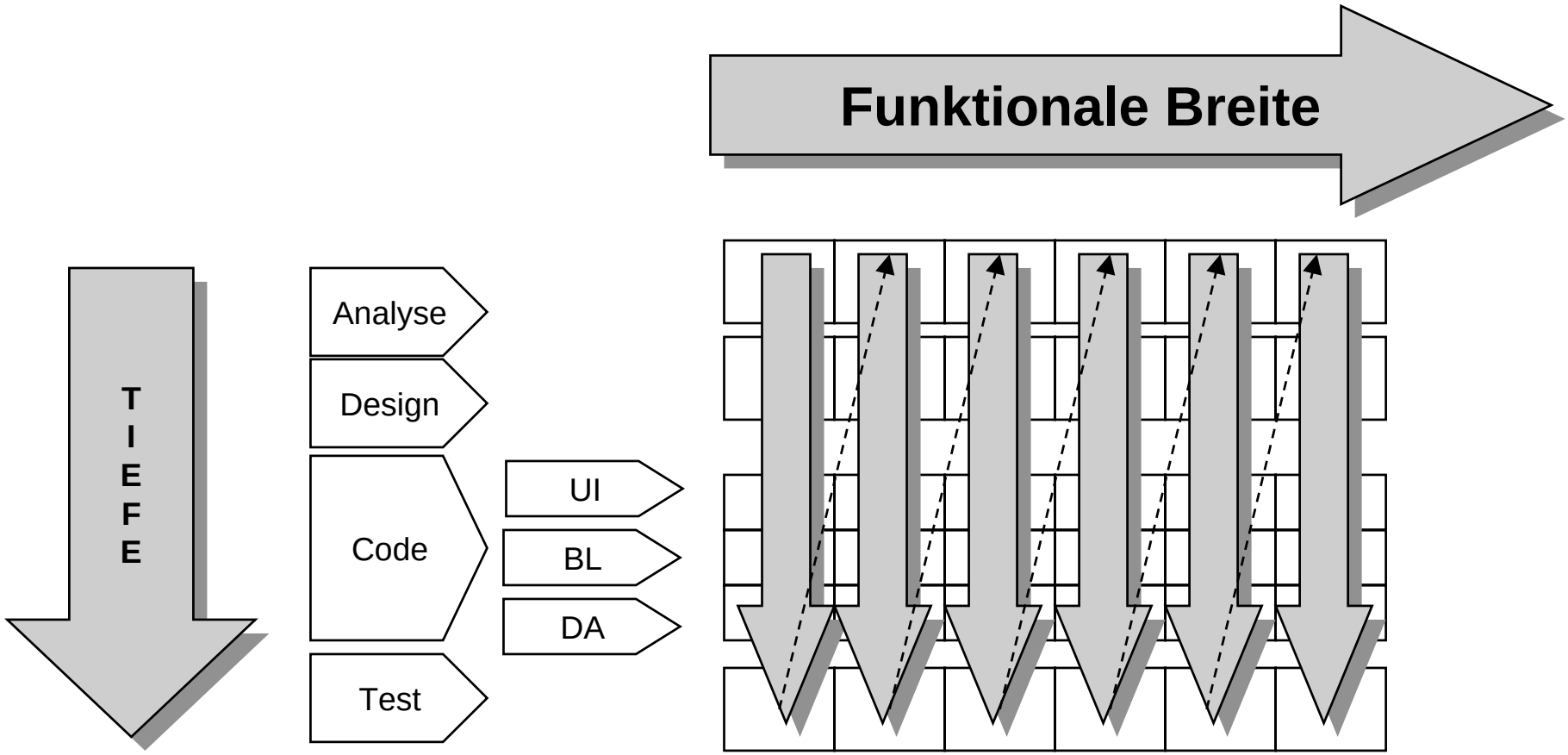
Idee: Aufteilen der Entwicklung in kurze Inkremente (3-6 Monate) ..



Überlappung
ist kein Zufall ..

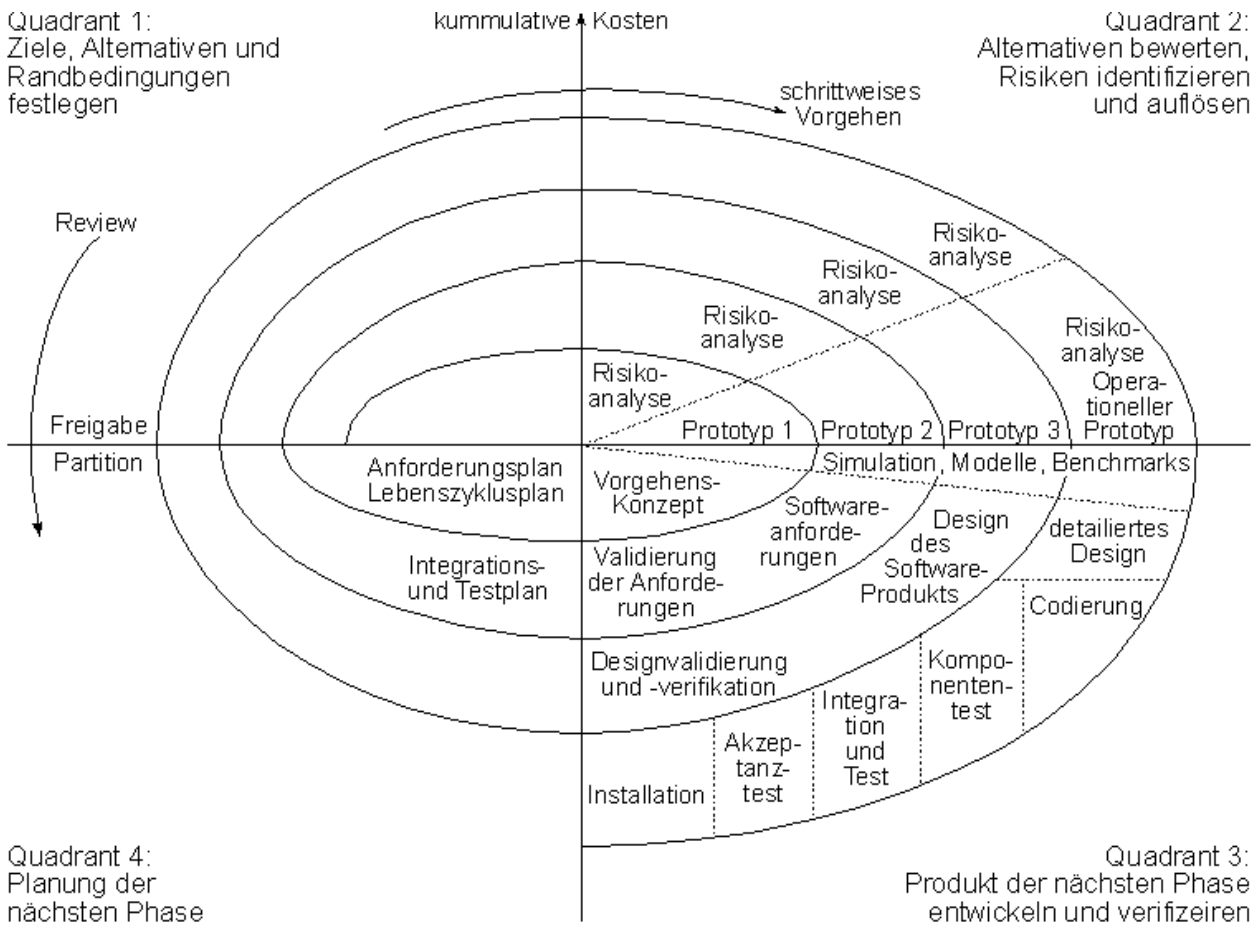
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell - Kästchenbelegungsstrategie



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell
Vorteile

- Man bekommt permanentes Feedback – auf Moving Targets kann reagiert werden
- Man kann pro Zyklus, wenn man möchte ein anderes Prozess und Teammodell wählen
- Fehler werden relativ schnell erkannt
- Man hat bessere Eingriffsmöglichkeiten als bei einem Wasserfall

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Spiralmodell
Nachteile

- Man braucht für das Spiralmodell ein besseres Management
- Für kleine Projekte viel Aufwand

Das wäre doch schon so gut gewesen, oder ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Haben wir eben einen Zusammenhang zwischen Architektur und Prozessen entdeckt?

Architekturthemen

Software-Entwicklung Lebenszyklus	Schlecht verstandene Anforderungen	Schlecht verstandene Architektur	Produziert hochverläßliche Systeme	Produziert Systeme mit guten Erweiterungsmöglichkeiten	enthält Risikomanagement	Kann sich an fixe Terminpläne halten	Hat geringen Overhead	Erlaubt Änderungen auch während der Halbzeit	Zeigt dem Kunden Fortschrittsergebnisse	Zeigt dem Projektmanagement Fortschrittsergebnisse	Kommt ohne erstklassige Manager oder erstklassige Entwickler aus	Optimierungskennziffer, kleiner ist besser
Pure Waterfall	5	5	1	1	5	2	5	5	5	2	2	28
Code-and-Fix	5	5	5	3	5	5	1	3	2	5	1	30
Spiral	1	1	1	1	1	2	2	2	1	1	4	15
Modified Waterfall	2	2	1	1	2	2	1	2	2	2	3	16
Evolutionary Prototyping	1	3	2	1	2	5	2	1	1	2	5	21
Staged Delivery	5	5	1	1	2	2	2	5	2	1	2	18
Evolutionary Delivery	2	5	2	1	2	2	2	2	1	1	2	15
Design-to-schedule	3	5	2	2	2	1	2	3	2	1	5	20
Design-To-Tools	2	3	3	5	4	1	2	1	1	1	2	20
Commercial Off-the-Shelf Software keine Angaben wurden auf 3 gesetzt	1	3	3	3	3	1	1	5	3	3	2	24

← Phoenix

← SLS

XP ähnlich

Optimierung auf Generaliverhältnisse
einschränkende Randbedingungen
Lockout-Kriterien

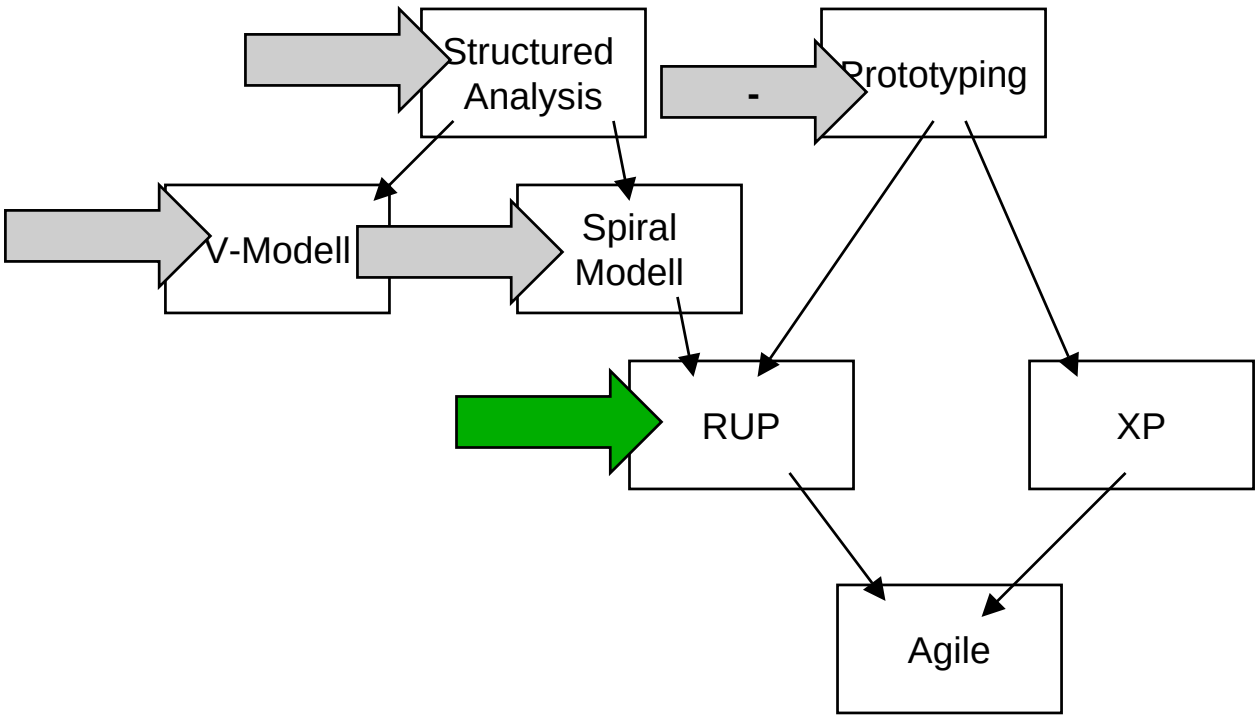
Bewertungen nach *Rapid Development*, Steve McConell, pp 156,157

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP - der Rational Unified Process („OO-Standardprozeß Heavy“)

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Historie RUP

OO Prozesse der „Tres Amigos“

- OMT [Rumbaugh]
- OOA/OOD [Coad/Yourdon]
- UseCaseDriven [Jacobson]
- ObjectOriented Design [Booch]

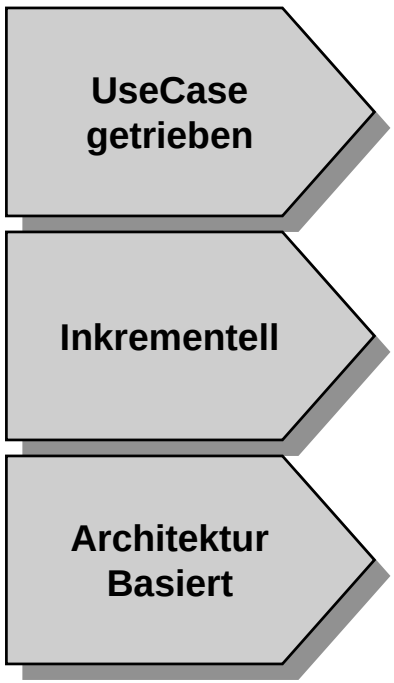
... Und als sie miteinander sprachen...

Rational Unified Process (RUP)
[Rumbaugh/Booch/Jacobson]

UNIFIED MODELING LANGUAGE

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP – use case getrieben, inkrementell, architekturbasiert



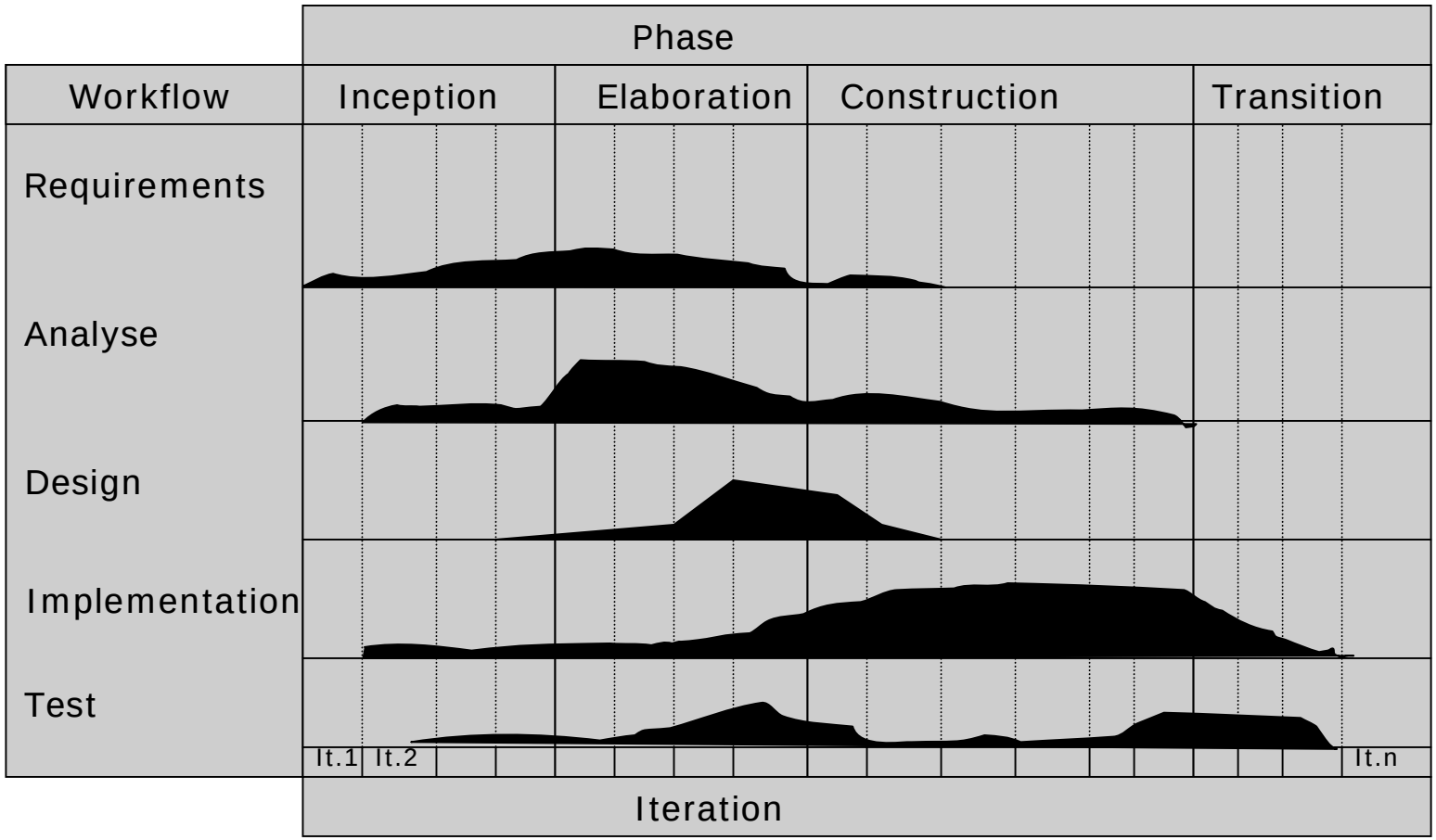
- **Use cases bilden die Basis für alle Phasen incl. Test**
- **Die Planung erfolgt nach UseCase-Paketen und Fertigungstiefe**

- **Das System wird in Iterationen errichtet**
- **Jede Iteration kann etwas mehr als die Vorgängeriteration**

- **Für Kritische UseCases wird eine Musterlösung erstellt**
- **Diese Lösung dient als „Schema F“ für die weiteren UseCases**
- **Diese Lösung heißt „Architektur“**

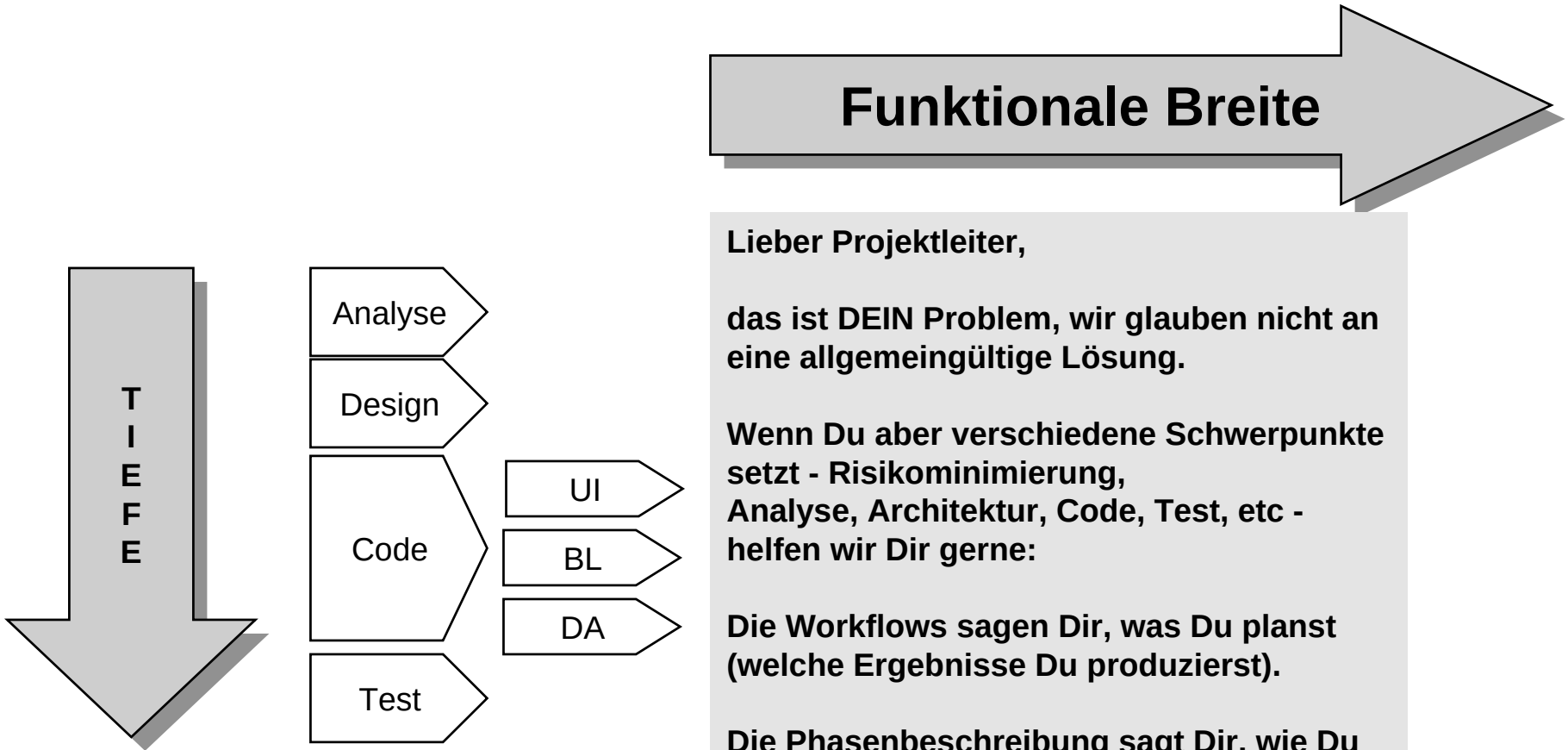
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP - Phasen und Workflows



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP - Kästchenbelegungstrategie



Lieber Projektleiter,

das ist DEIN Problem, wir glauben nicht an eine allgemeingültige Lösung.

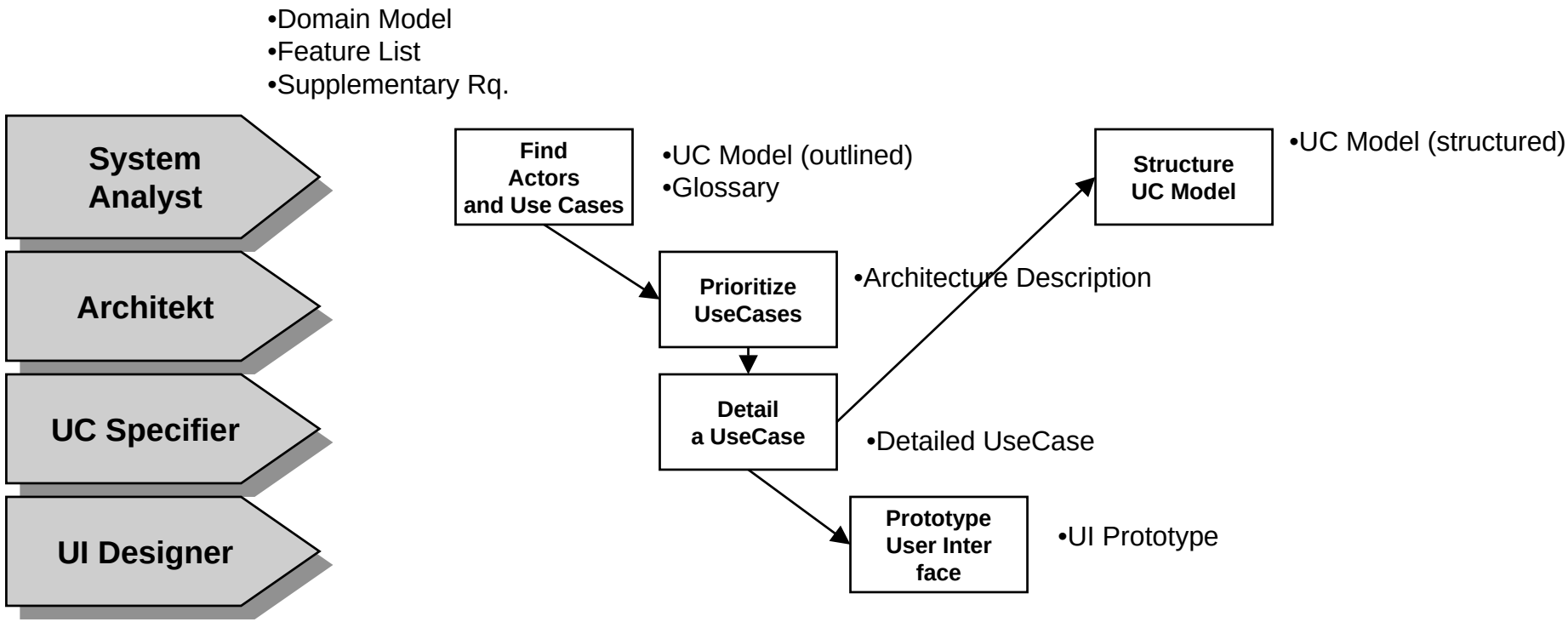
Wenn Du aber verschiedene Schwerpunkte setzt - Risikominimierung, Analyse, Architektur, Code, Test, etc - helfen wir Dir gerne:

Die Workflows sagen Dir, was Du planst (welche Ergebnisse Du produzierst).

Die Phasenbeschreibung sagt Dir, wie Du planst (worauf Du achten mußt).

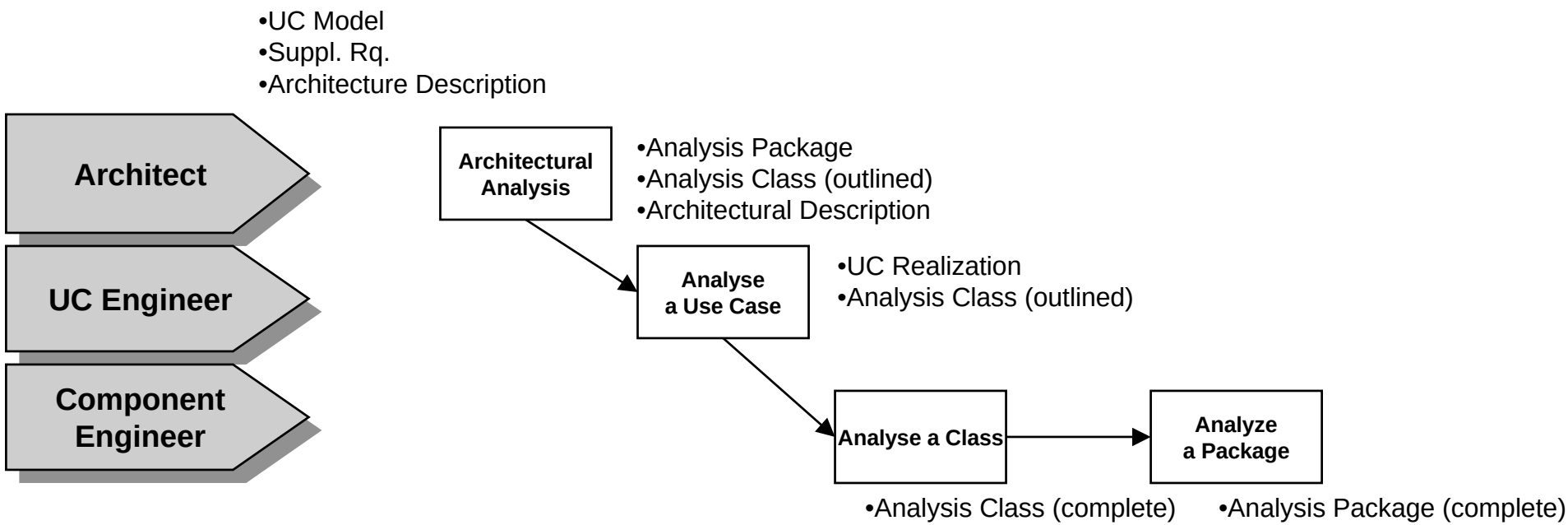
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP -Requirements WF



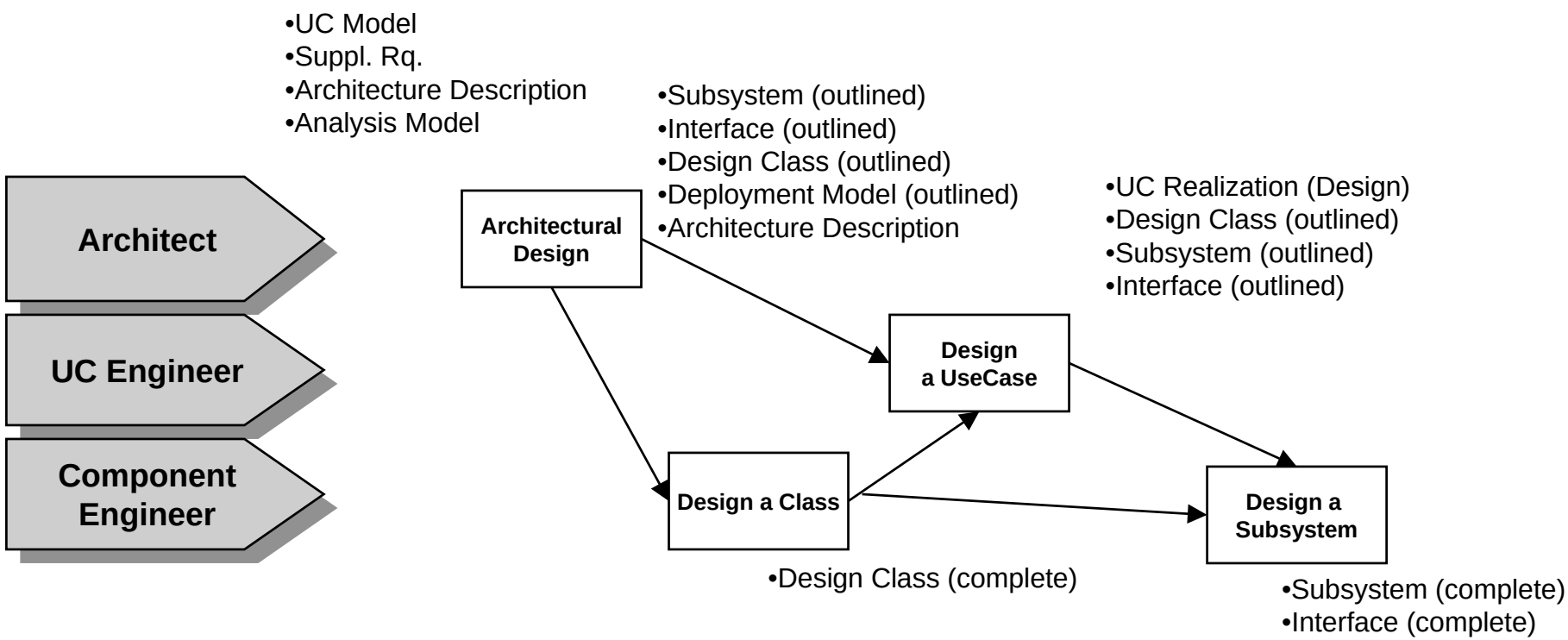
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP -Analysis WF



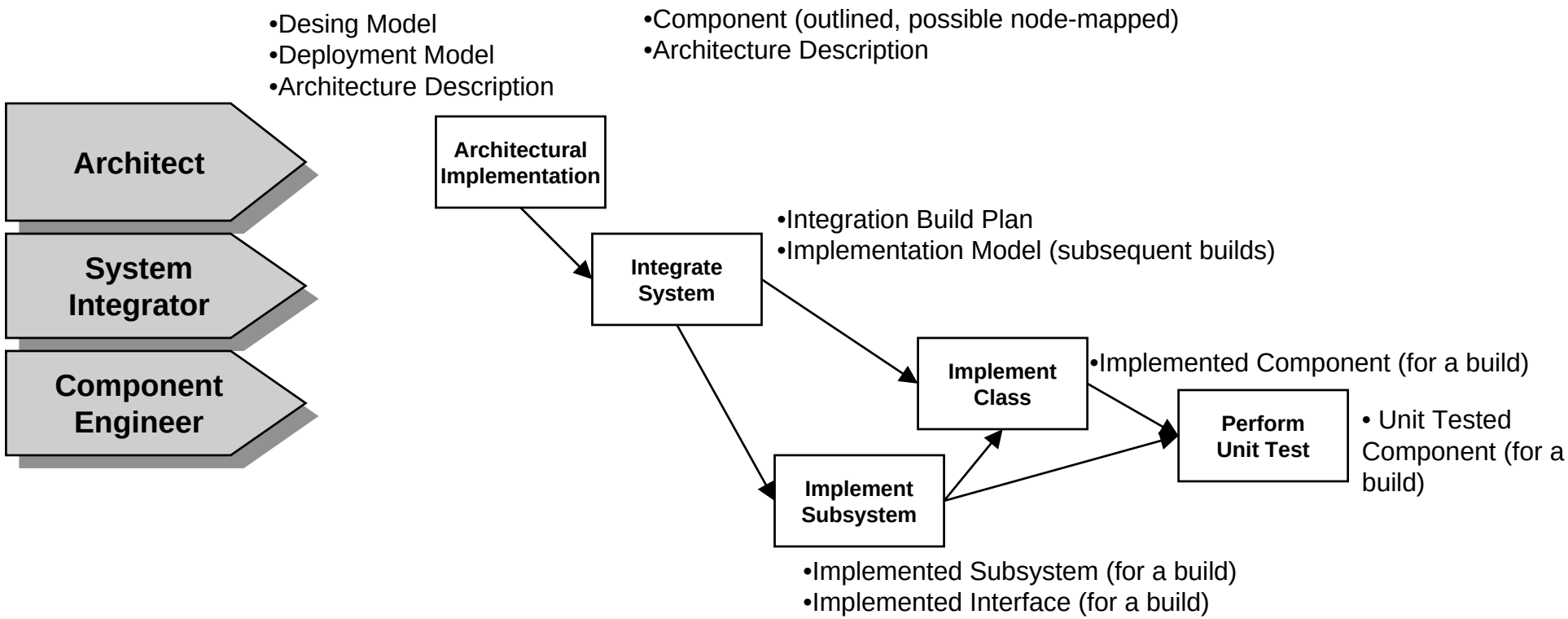
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP -Design WF



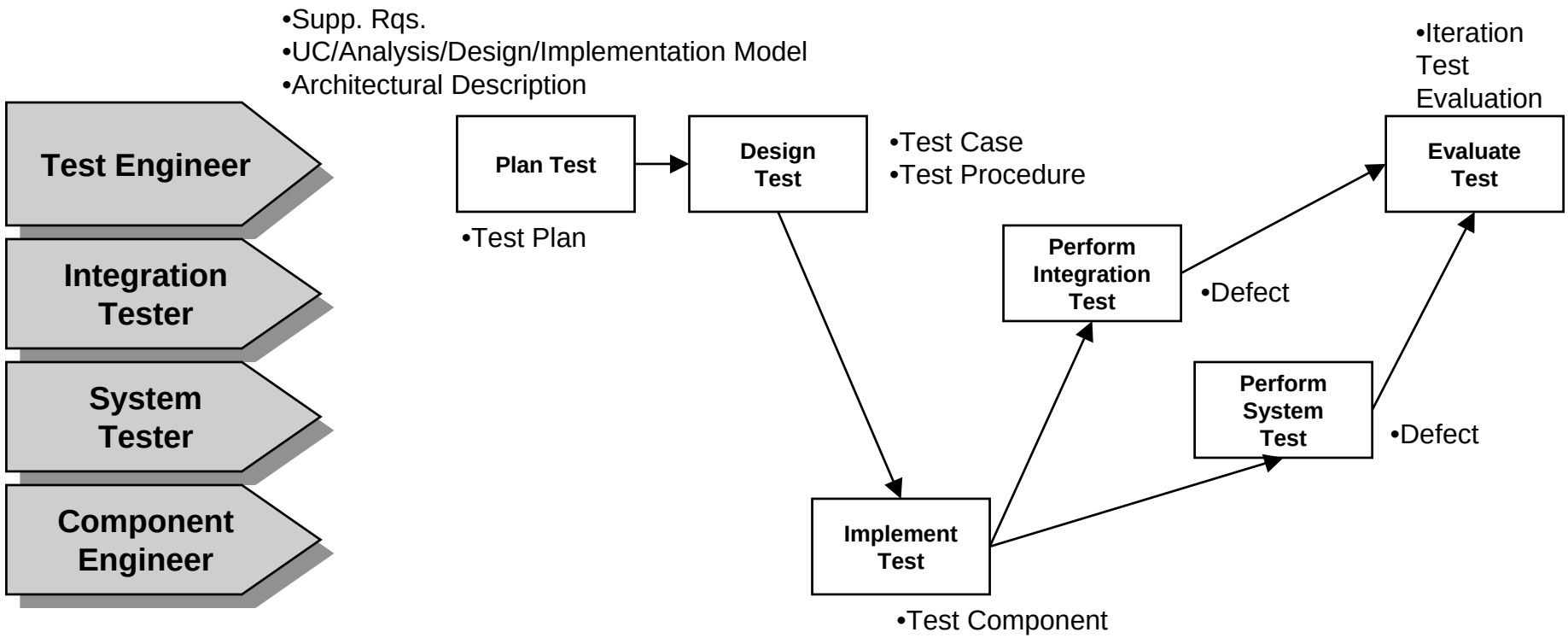
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP - Implementation WF



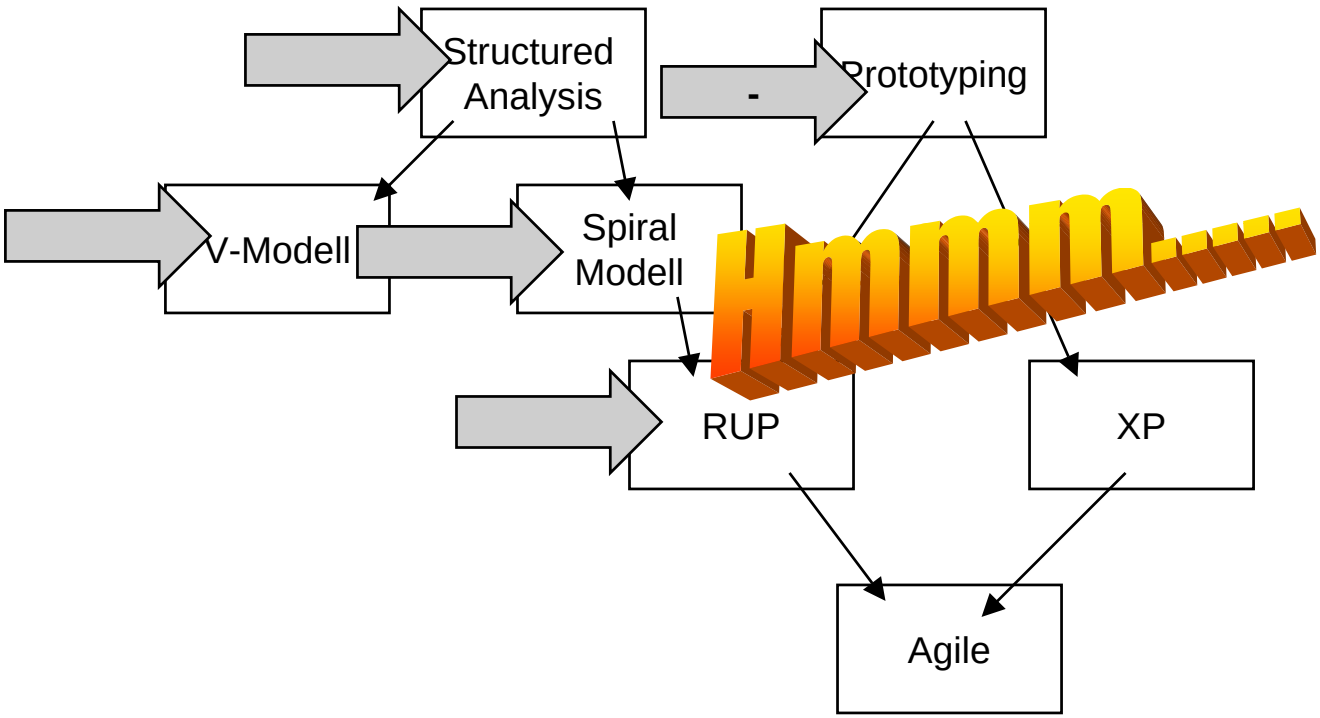
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

RUP - Test WF



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

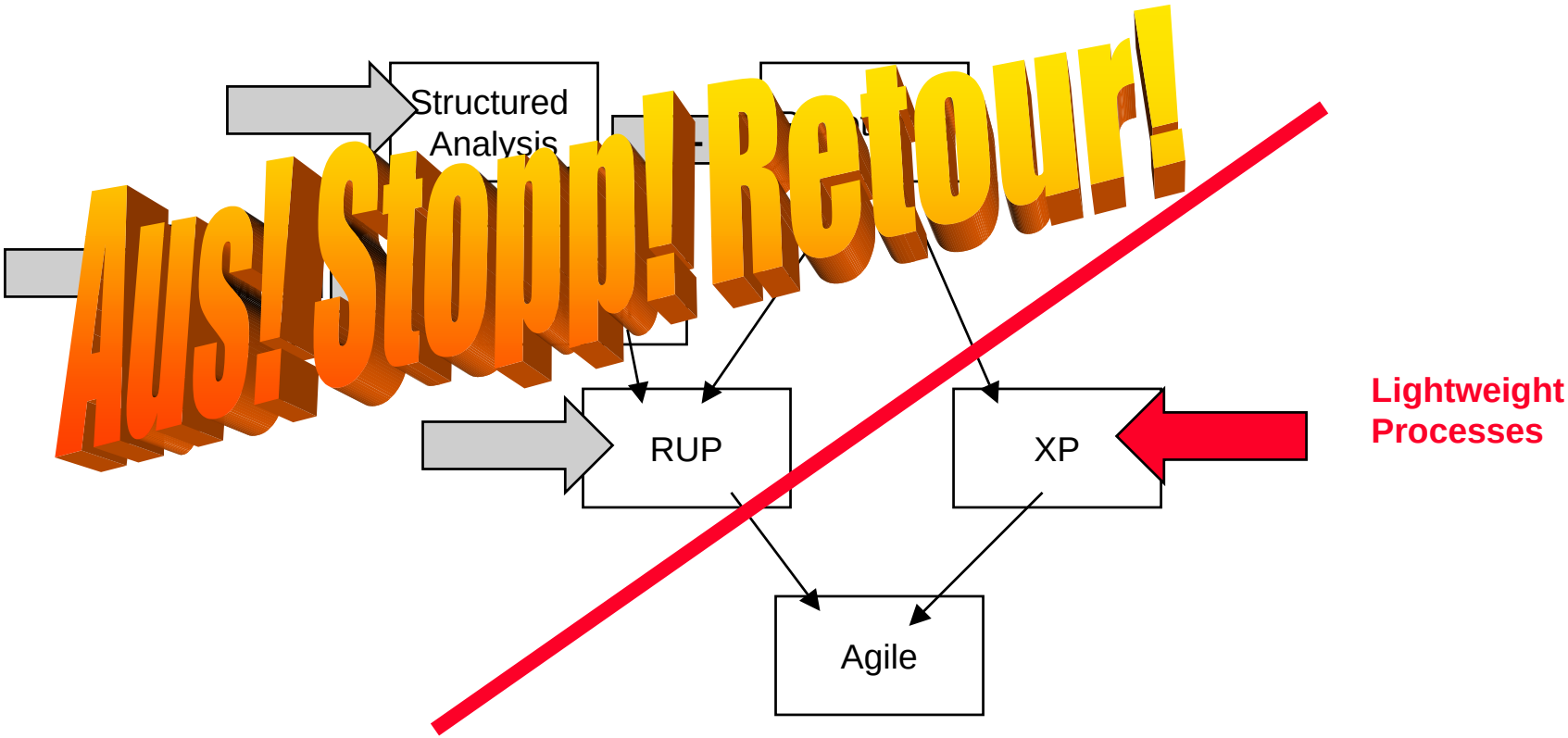
Grundprinzip traditioneller Entwicklungsmethoden

- Unterdrückung von Individualismus
- Tayloristisches Management - Management by Fear
- Misstrauen
- Repeatable Processes (ISO 9000, Capability Maturity Model

- **<u>Das Individuum zählt nicht!</u>**
- **<u>= MACHT KEINEN SPASS</u>**

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

eXtreme Programming
als eine mögliche agile Methode

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wesentliche Richtungen

- XP stellt die **Programmierung** in den Vordergrund – der Code ist das, wofür bezahlt wird
- XP nimmt Moving Targets als den Normalfall an. Änderungen werden nicht bekämpft – **embrace change**
- Bei XP steht im Verhältnis Auftraggeber (AG) und Auftragnehmer **Vertrauen** im Vordergrund ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Die einzelnen Praktiken von XP sind nicht neu ..

- Code Reviews sind gut
  - also werden Sie dauernd gemacht
  - **Pair Programming**
  - und es gibt informelle **Coding Standards**
- permanentes Testen ist gut
  - also werden die Testfälle ständig auf dem Stand gehalten und ausgeführt
  - JUnit, SUnit, Use Cases zugleich als Testscripts zu verwenden -> **Unit Testing.** **Und Testfälle werden gebaut, bevor codiert wird.**
- Integrationstests sind wichtig
  - deshalb wird permanent integriert und wieder getestet. **Continuous Integration**

Das macht auch der einsame Shareware Programmierer.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Die einzelnen Praktiken von XP sind nicht neu ..

- Design ist wichtig
  - deshalb wird es während des Codierens durch **Refactoring** ständig verbessert
- Einfachheit ist gut
  - als wir das einfachste gemacht haben, was in diesem Moment noch funktioniert hat – ich glaube das ja nicht☺ (HH)
  - „**do the simplest thing that might possibly work**“
- Es geht allerdings nicht ohne Architektur und Architekturverständnis!

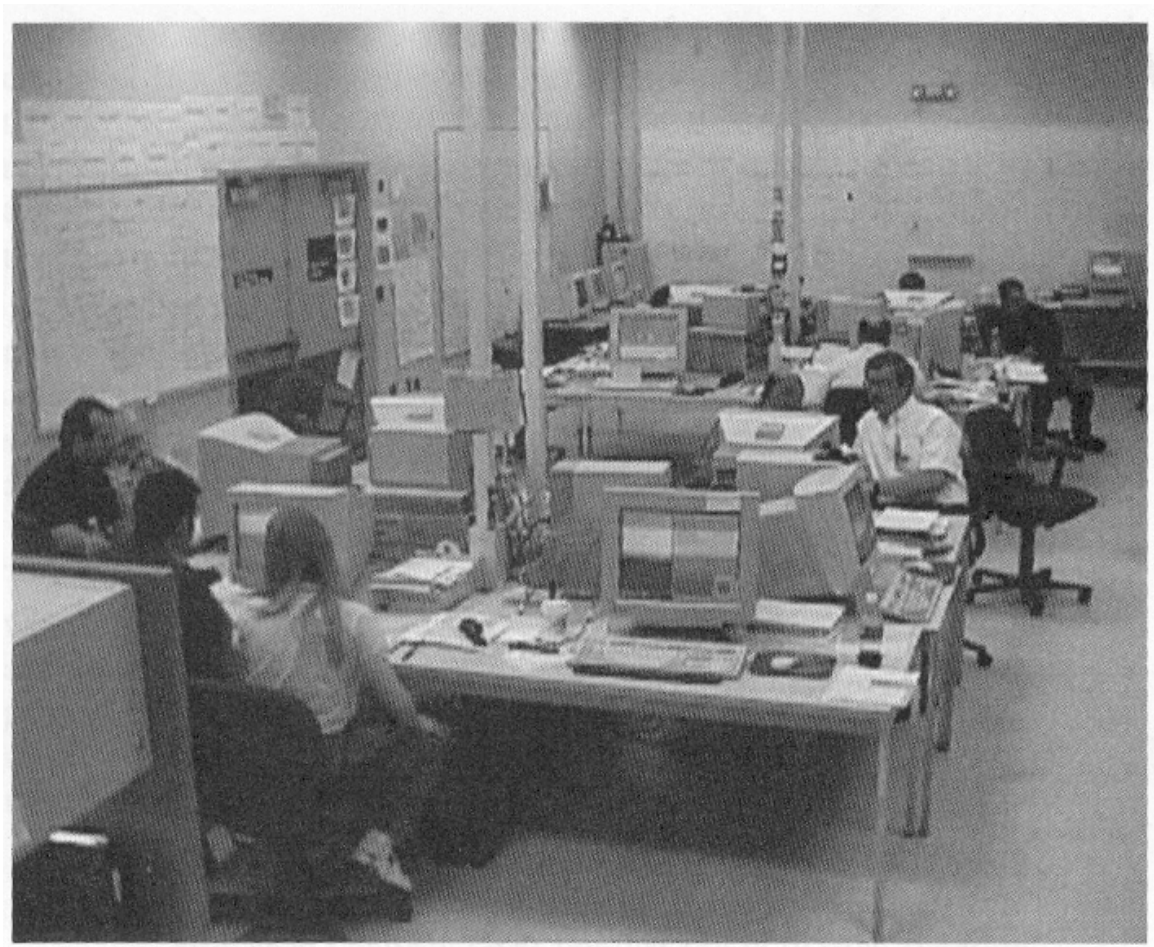
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Die einzelnen Praktiken von XP sind nicht neu ..

- Kurze Iterationen sind wichtig
  - also werden die Iterationen extrem kurz gemacht
  - 10 Tage, dann wieder „Planning Game“
- Feedback durch den Kunden ist wichtig
  - also ist der Kunden im selben Raum
  - **on site customer**
- **40 Stunden Woche,** weil die Leute sonst ausbrennen, wie man in der EDV überall bewundern kann ..

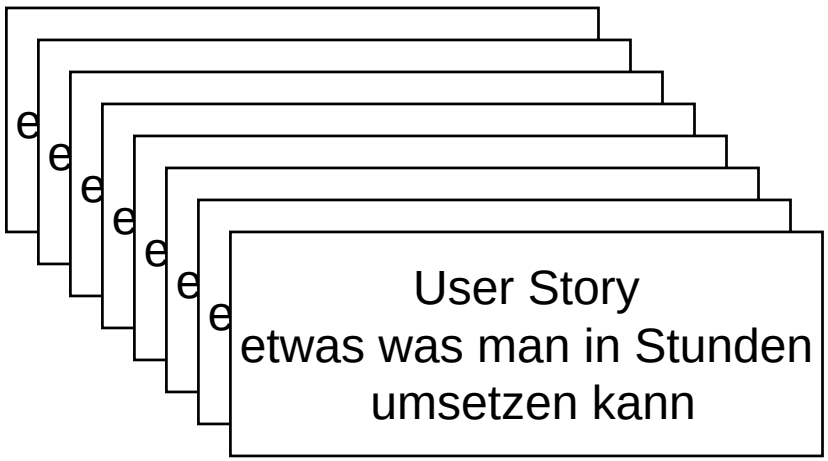
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Beispiel: Raumaufbau ...



Vorlesung „Software Engineering für große, betriebliche Informationssysteme“,
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

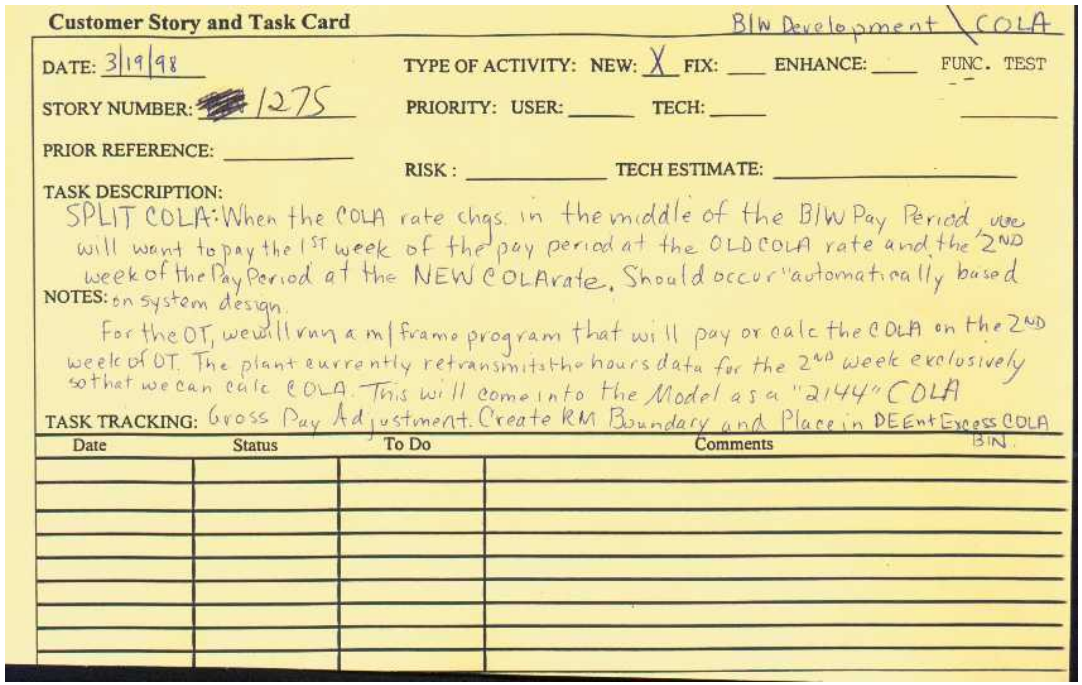
Wie funktioniert das Planning Game?



- Team sammelt Stories
- diese werden für das nächste Inkrement „priorisiert“
- und dann abgearbeitet
- wenn sich dabei „Planungsfehler“ herausstellen ist das o.k. und wird sofort korrigiert (durch splitten der Story oder Besprechung im Team)
- nach einem Inkrement setzt man sich zusammen und macht das ganze von vorne ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Beispiel für eine Story Card



Customer Story and Task Card B/W Development \ COLA

DATE: 3/19/98 TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: [redacted] 1275 PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs. in the middle of the B/W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur "automatically based on system design.

NOTES:
For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEnt Excess COLA BIN.

Date	Status	To Do	Comments

Quelle: http://www.xprogramming.com/images/StoryCard.jpg

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wie funktioniert permanentes Testen?

- man schreibt erst die Testfälle und dann den Code
- die Testfälle können bei Smalltalk zum Beispiel mit den Use Cases identisch sein
- oder man schreibt pro Klasse Testfälle
- damit hat man das Interface meist definiert ... Und tut auch etwas für gutes Design – wenn man den Code des Interface gut lesen kann ist er auch meist gut ..
- JUnit o.ä. wird eingesetzt, um Testfälle permanent laufen lassen zu können ...

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Was bitte ist Refactoring? Bespiel 1

Replace Magic Number with Symbolic Constant

You have a literal number with a particular meaning.

Create a constant, name it after the meaning, and replace the number with it.

ace Magic
mber with
Symbolic
Constant

```
double potentialEnergy(double mass, double height) {
    return mass * 9.81 * height;
}
```

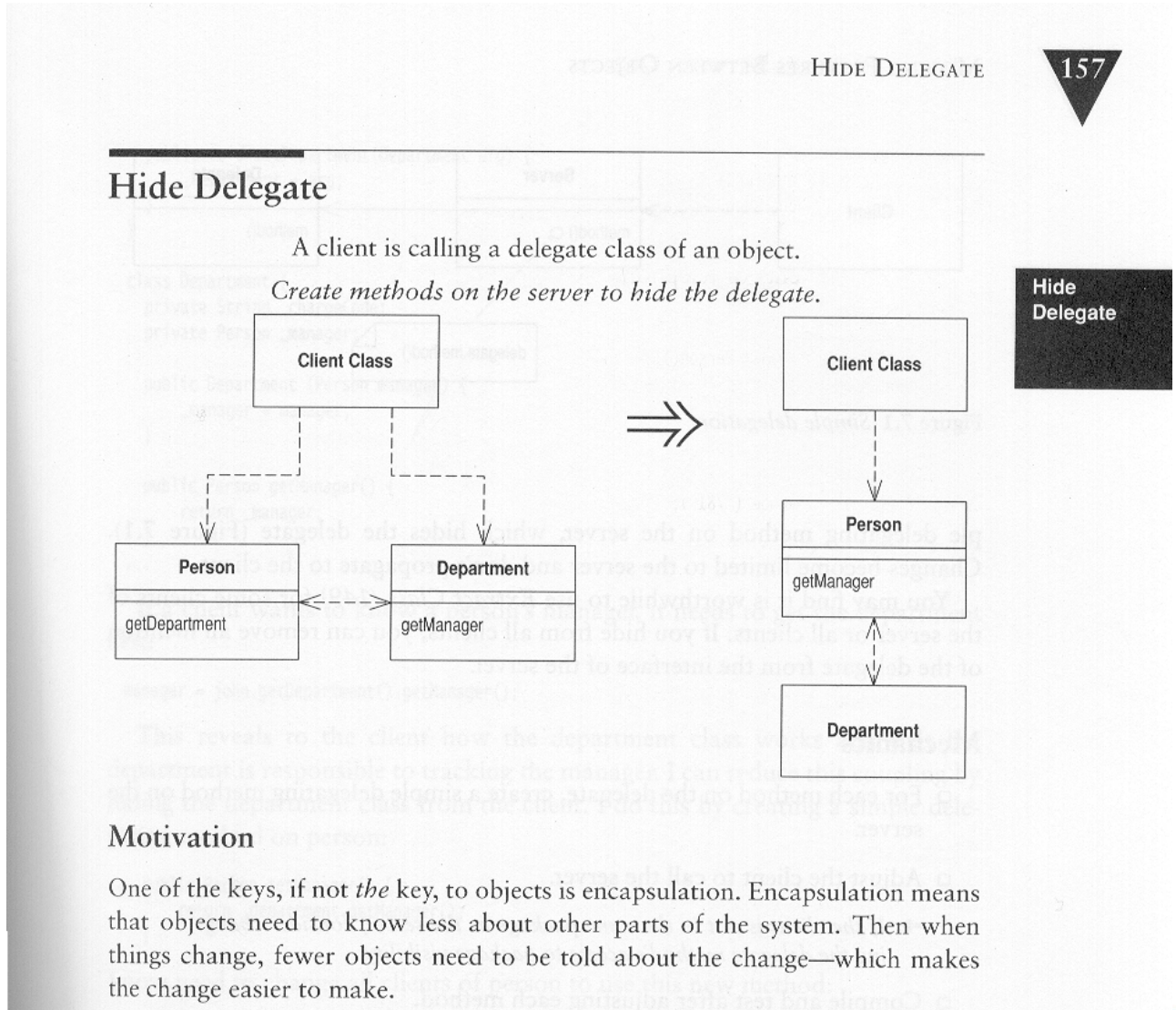
⇓

```
double potentialEnergy(double mass, double height) {
    return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Quelle: [Fowl1999]

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Was bitte ist Refactoring? Beispiel 2:



Hide Delegate

A client is calling a delegate class of an object.

Create methods on the server to hide the delegate.

Motivation

One of the keys, if not *the* key, to objects is encapsulation. Encapsulation means that objects need to know less about other parts of the system. Then when things change, fewer objects need to be told about the change—which makes the change easier to make.

Quelle:
[Fowl1999]

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Refactoring

- Refactoring wurde nicht von Martin Fowler erfunden sondern von William Opdyke, der bei Ralph Johnson gearbeitet hat
- Refactorings sind Pattern (Muster) mit vorher/nachher Zuständen, wie man Code verbessern kann
- Refactoring funktioniert besser, wenn man ein Tool hat, das Refactoring unterstützt. So was gab‘s erst für Smalltalk - jetzt auch für Java
- Refactoring setzt voraus dass man schnell testen kann, was man getan hat

- Schauen Sie sich einmal an, wie man mit eclipse Refactoring betreiben kann. Gamma macht das während eines Vortrages mit der linken Hand.

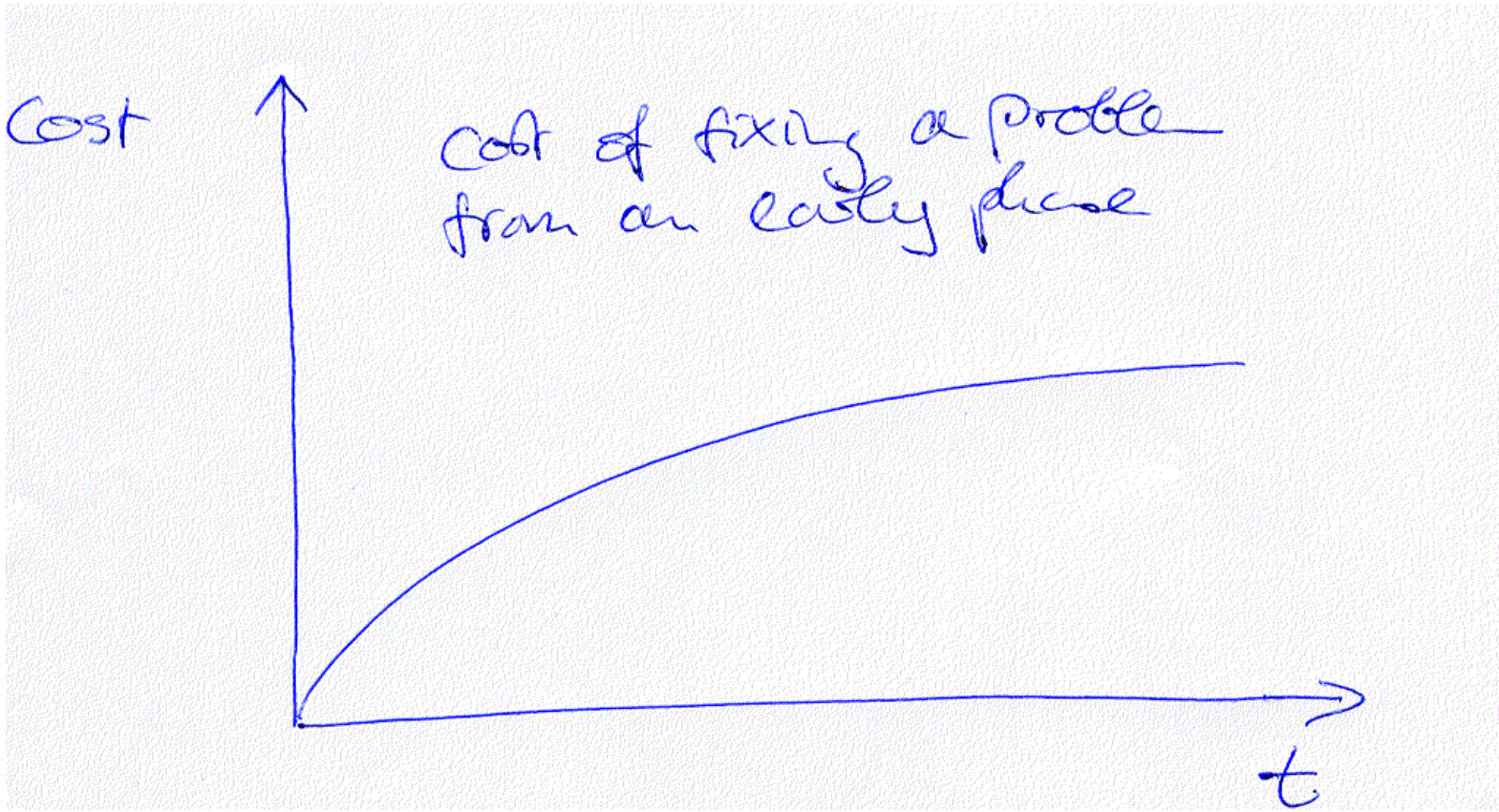
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wie hängen Refactoring und Unit Testing zusammen

- mittels Unit Testing kann man feststellen, welche Testcases durch die Änderung gebrochen werden
- Das ist keine 100% Garantie aber bei einer ausreichenden Anzahl Testfälle eine ausreichende praktische Sicherheit

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Smalltalk, Refactoring, Unit Testing ergeben (hoffentlich) ...



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Caveats

- XP is like teen-age sex – lots of talk about it but hardly anyone is doing it!
  - Schöne Beispiel: Auf der OOPSLA 2000 wurde ein 2 Wochen Internet Projekt als XP verkauft
- XP ist nur wenige Jahre alt. Es wurde imChrysler C3 Project erfunden und praktiziert …
  - keiner weiß, ob es für große Projekte skaliert. Aber es gibt erste seriöse Angaben, dass es CMM-3 tauglich ist.
- XP war ein Hype
  - Bis jeder darunter Windows verstanden hat.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

XP unterliegt gewissen Prämissen

- kleines Entwicklungsteam (2-10)
- gute Leute – die nicht anfangen zu hacken
- die Kosten von Änderungen sind gering
  - Grundlage Refactoring, Refactoring Browser, schnelle Recompiles oder interpretierte Sprache (Smalltalk)
- die Software-Entwicklungsumgebung erlaubt permanente Tests
- Das Design ist einfach und sauber – keine gigantischen Frameworks ..

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Vorsicht Hype

- Manche benutzen XP als Ausrede um nicht zu dokumentieren ..
  - im Chrysler C3 Projekt wurde nicht dokumentiert
  - das hat später durch Wechsel von Teammitgliedern zu Problemen geführt
  - ausserdem wurde das Projekt inzwischen abgebrochen - angeblich aus „politschen Gründen“

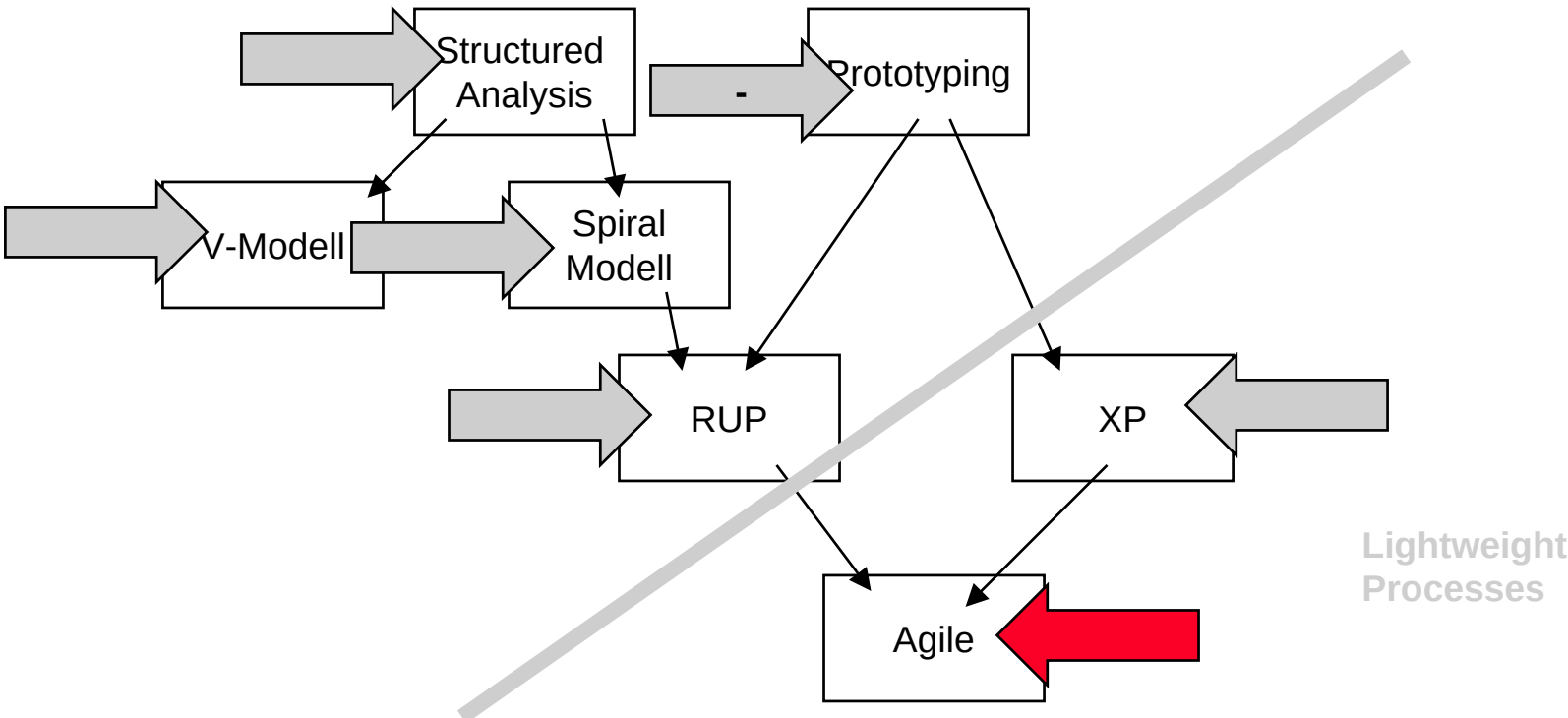
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Vorsicht Hype

- Manche sagen, sie machen XP
  - nur leider ist der Kunden nicht greifbar – nicht im selben Raum
  - damit kann es nicht mehr funktionieren und bleibt dann lediglich eine Ausrede für schlechte Architektur und mangelnde Dokumentation

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fragwürdiger Stammbaum der Entwicklungsprozesse



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse

oder was kann und sollte man in großen Projekten von XP einsetzen

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Wir haben gesehen: Projekte haben sich seit den 70er Jahren stark geändert

70er

- Abbildung stabiler Geschäftsprozesse
- Informatik nicht unbedingt Waffe im Wettbewerb
- Wenige technische Möglichkeiten. Lange Technologiezyklen
- Moving Targets waren ein Indikator für schlechtes Management

Heute

- es ist die Regel, dass sich Geschäftsprozess schnell ändern
- Informatik wird als Waffe im Wettbewerb gesehen
- Eine Flut von Technologien, mit Innovationszyklen kleiner 2 Jahre
- Moving Targets sind der Normalfall

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein paar Grundprinzipien agiler Entwicklung

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

[Cobu2002]

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein paar Grundprinzipien agiler Entwicklung

- Vertrauen zählt
- Der Kunde zählt
  - oder haben Sie schon mal einen Konflikt mit einem Kunden dauerhaft gewonnen
- Das Individuum zählt
  - seine Kommunikationsfähigkeit
  - sein Können als Softwareingenieur
  - und das Wissen als Fachmann in der Domain

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

und schon wieder – wieviel Sinn macht hier ein Prozesshandbuch?

- die wirklich guten Fahrer benutzen kein Prozesshandbuch
- auch die wirklich guten fliegen mal von der Strasse, weil sie kontrolliert Risiken eingehen
- das Prozesshandbuch hilft den Anfängern besser zu werden
- für Meisterschaft muß man üben, üben, üben
- Kent Beck ist sowas wie ein Schumi der Smalltalk-Programmierung

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse: Was von XP wird z.B. bei uns viel verwendet?

- Früh Feedback abholen
  - niemand darf mehr als 3 oder max. 6 Monate „vor sich hinarbeiten“ ohne Software zu zeigen!
- Frühes Test, viel Testen
  - kann man in fast jeden Prozess einbauen – sogar bei Host Entwicklung
- Do the simplest thing
  - ist eine gute Hilfe um „Komplexdenker“ einzubremsen!
  - Warum machst Du das – tust Du damit etwas für den Kunden oder nur für Dein Ego als Programmierer?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse: Was war z.B. bei uns schwer umzusetzen

- 40 Stunden Woche
  - es gibt immer noch ziemlich schädliche, aber weit verbreitete Death March Kulturen
- Vertrauen
  - weil Manager den Mitarbeitern zu wenig vertrauen
  - und implizit davon ausgehen, dass diese „Faule Säcke“ sind, die sich nicht richtig anstrengen, wenn man sie nicht ordentlich unter Druck setzt
  - weil Festpreissituationen das institutionalisierte Misstrauen sind
  - weil man dafür gute Leute braucht – wieviele Kent Becks kennen Sie?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse: Was war z.B. bei uns schwer umzusetzen

- on site customer
  - weil viele Unternehmen keine Projektkultur haben
  - und Mitarbeiter der Fachbreiche nicht die Bindung verlieren wollen (Das Projekt als Karriereknick!)
- Refactoring
  - weil das hohe Ansprüche an die technische Umgebung stellt und zum Beispiel mit COBOL so gut wie nicht zu machen ist ...
- Coding Standards
  - verwässern, wenn man nicht stark „hinterher ist“

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse: Was war z.B. bei uns schwer umzusetzen

- Metaphern
  - erfordern Liebe zum Thema, großes Wissen und Liebe zum Job.
  - habe ich noch wenig gesehen

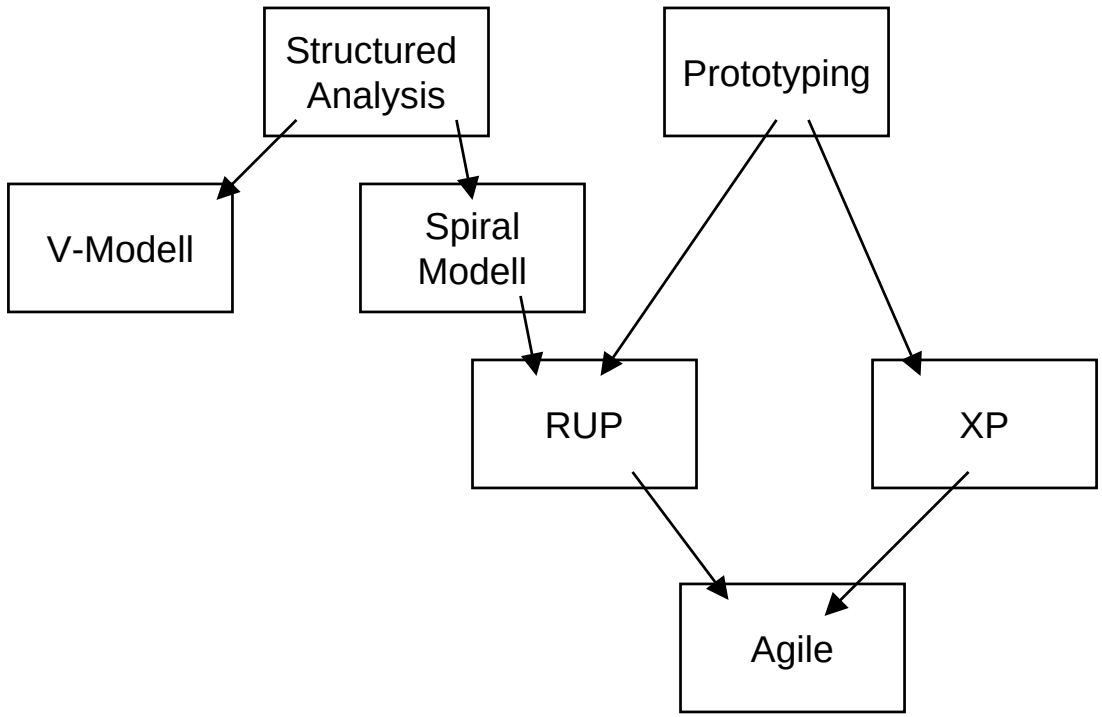
© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Agile Prozesse: Was von XP haben wir noch wenig probiert

- Pair Programming
  - nicht jedermanns Sache
- Continous Integration
  - ist eine starke Disziplinfrage

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fertig? ...



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Von der Theorie zur Praxis – der SEP (Softwareentwicklungsprozeß der Generali VIS Informatik GmbH)

Oder: wie agil kann man in einem großen Unternehmen agieren?

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Der Software-Entwicklungsprozeß ist kein Isolani*)

- Der Prozeß muss auch in mehreren, parallelen Projekten gelebt werden können.
- Anforderungen kommen „irgendwann“ daher, nur nicht zu Projektbeginn.
- Projekte verändern eine Anwendungslandschaft, die aus Anwendungen besteht. – Jedes Projekt resultiert in Veränderungen in mehreren Anwendungen.
- Daher ist der SEP eingerahmt durch den A
  Auslieferungsprozess

Vis-Prozess

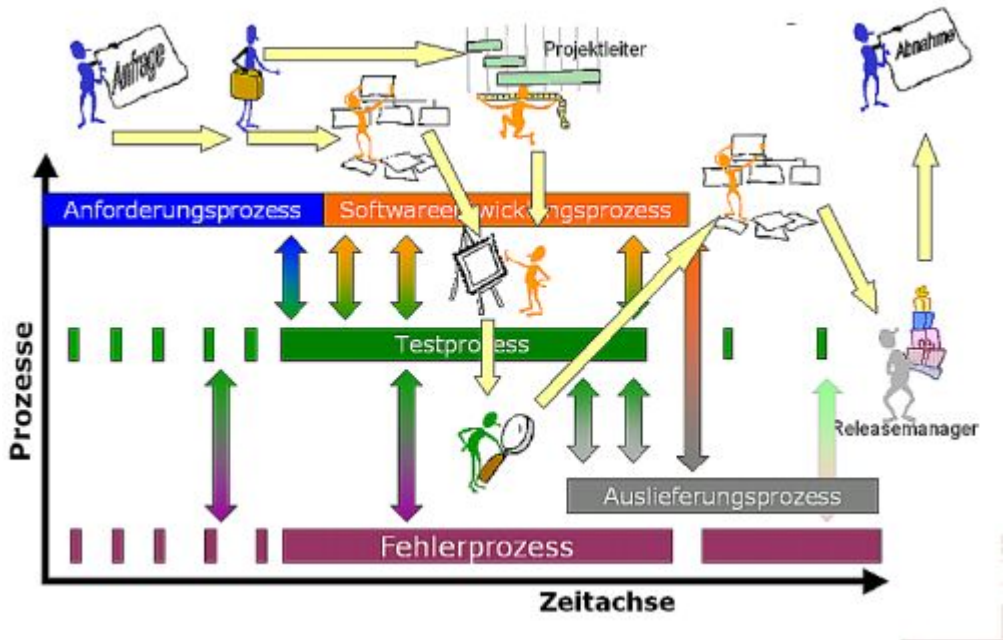
- _____

Isolani .. Isolierter Bauer, der nicht durch einen Anderen Bauern gedeckt werden kann.
Begriff aus unserem ursprünglichen Schachprogramm

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Der Software-Entwicklungsprozess ist ein bisschen RUPpig!

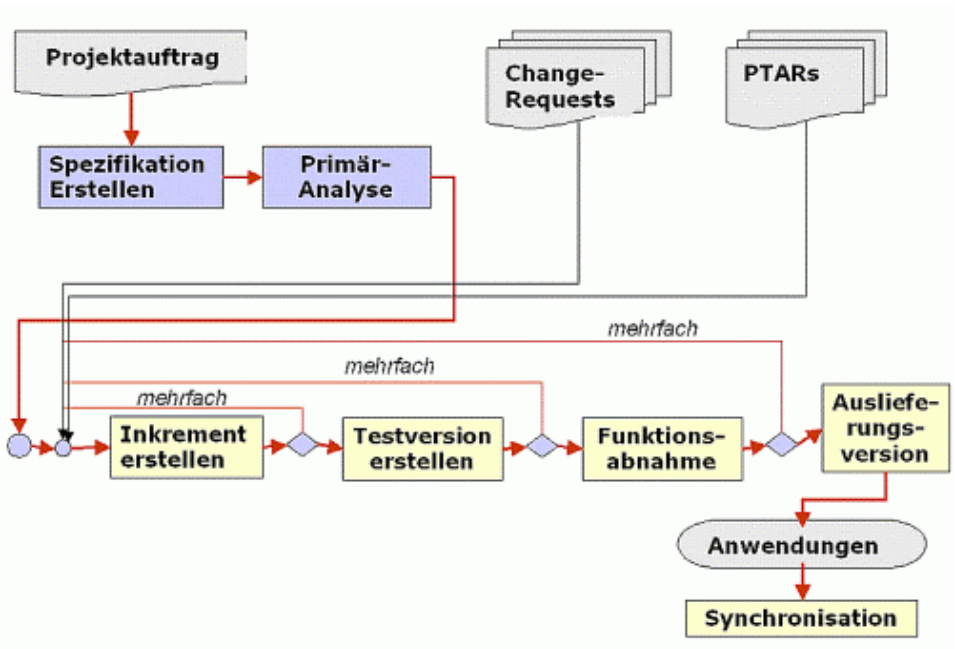
- Die einzelnen Prozesse und damit Tätigkeiten spielen sich in Phasen ab, die n i c h t nur einer Tätigkeit zugeordnet sind.
- Der Testprozeß zieht sich als Ko-Prozeß zum SEP über die gesamte Projektdauer.



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

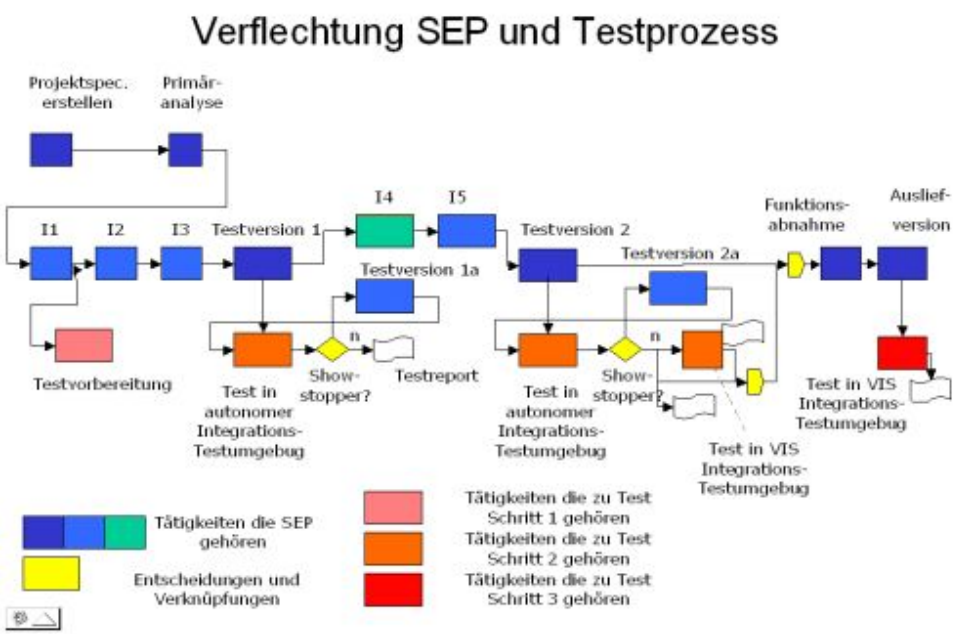
Der Software-Entwicklungsprozess ist ziemlich simpel

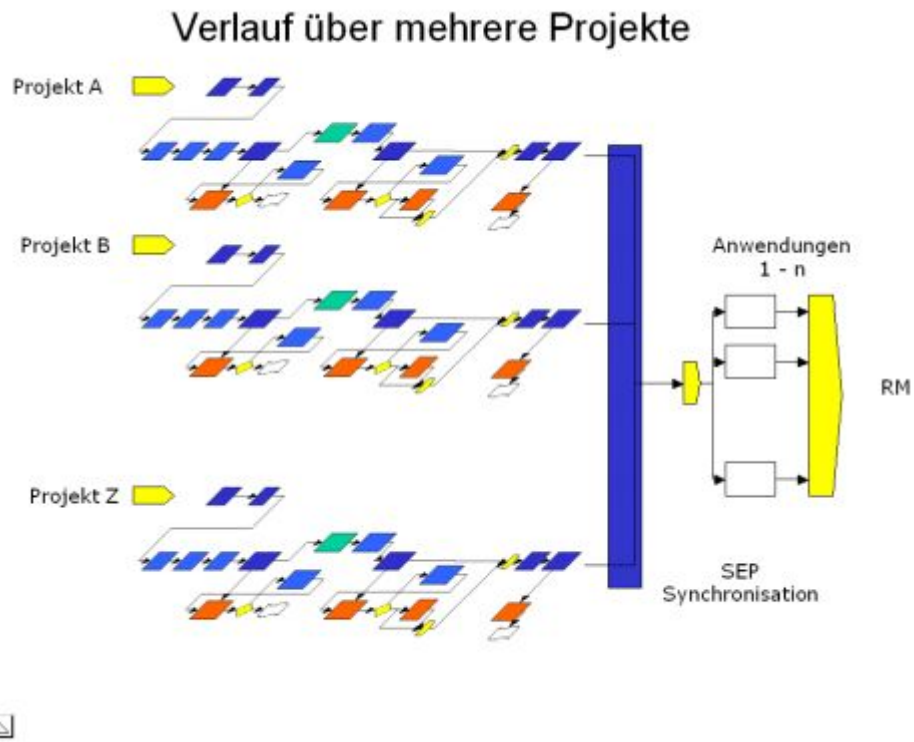
- Er besteht aus 7 diskreten Einheiten und ist auf das Unternehmen taylorisiert.
  - Primäranalyse fasst zwei Schritte zusammen: Analyse und Design
  - Das funktioniert bei uns, weil auch die fachliche Analyse im wesentlichen von den Programmierern niedergeschrieben wird.



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Ein paar Einzelheiten muss man allerdings bedenken





Mit Test zusammen ist es komplizierter, und mit vielen Projekten und Anwendungen ist es noch komplizierter.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Der Software-Entwicklungsprozeß schreibt nicht vor, wie man etwas tut, sondern was man erzeugt

- Der Output des Schrittes Inkrement liest sich dann so:

Inkrement Output

Site Map

SEP
Spez.erst.
Prim.Analyse
Inkrement
Testversion
Auslief.Vers.
Funktionsabn.
Synchronisation
Schritt-Zusatzinfo

Input Inkr
Output Inkr
Rollen Inkr
Besonderes Inkr

Eine Beschreibung zu den Dokumenten findet sich dort.		Dieser Schritt des SEPs erzeugt Code und stellt folgende Ergebnisse bereit:
SEP-3a.1 verwendet in	C/U-Test	**Lauffähige Module,** die im Entwickler-Gesamtbuild mit erfolgreichem Code/Unit-test durchlaufen und in cvs versioniert sind
SEP-1a.4 verwendet in		**Laufzettel**: überarbeitet (was zum Zeitpunkt der Inkremententwicklung an neuen Erkenntnissen dazugekommen ist.
SEP-2a.2 wird verwendet		**Funktions-Backlog:** aktualisiert. Insbesonders müssen hier neu hinzugekommene CRs aus dem Projekt eingetragen werden. Solche CRs werden dann gestellt, wenn dem Inkrement eine Testversion vorangegangen ist und ein gefundener PTAR als CR klassifiziert wird.
SEP-3a.3 gültig bis zum nächsten Inkrementende		**Liste der realisierten Funktionalitäten**: = Erfolgsmeldung
SEP-3a.4 gültig bis zum nächsten Inkrementende		**Liste der nicht - wie ursprünglich geplant - realisierten Funktionalitäten**: die hier angeführten Punkte sind in kommende Inkremente einzuplanen und können daher den Zeitplan des Projektes umwerfen.
	Legende C/U-Test verwendet in Grundlage	*C/U-test: Code-Unit-Tests sind automatisierte Tests, welche von den Entwicklern programmiert und gestartet werden.* *verwendet in: das Dokument lebt und wird in weiteren Schritten ergänzt und bearbeitet.* *Grundlage: das Dokument klärt Grundsatzfragen und stellt eine Referenz für neu zu lösende technische und fachliche Probleme dar.*

Zurück zu Inkrement entwickeln

Was ist der SEP? [SEP] [Übersicht] [SEP-Ablauf] [Werkzeuge + Dok] [Methoden] [Kleingedrucktes] [FAQ]

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

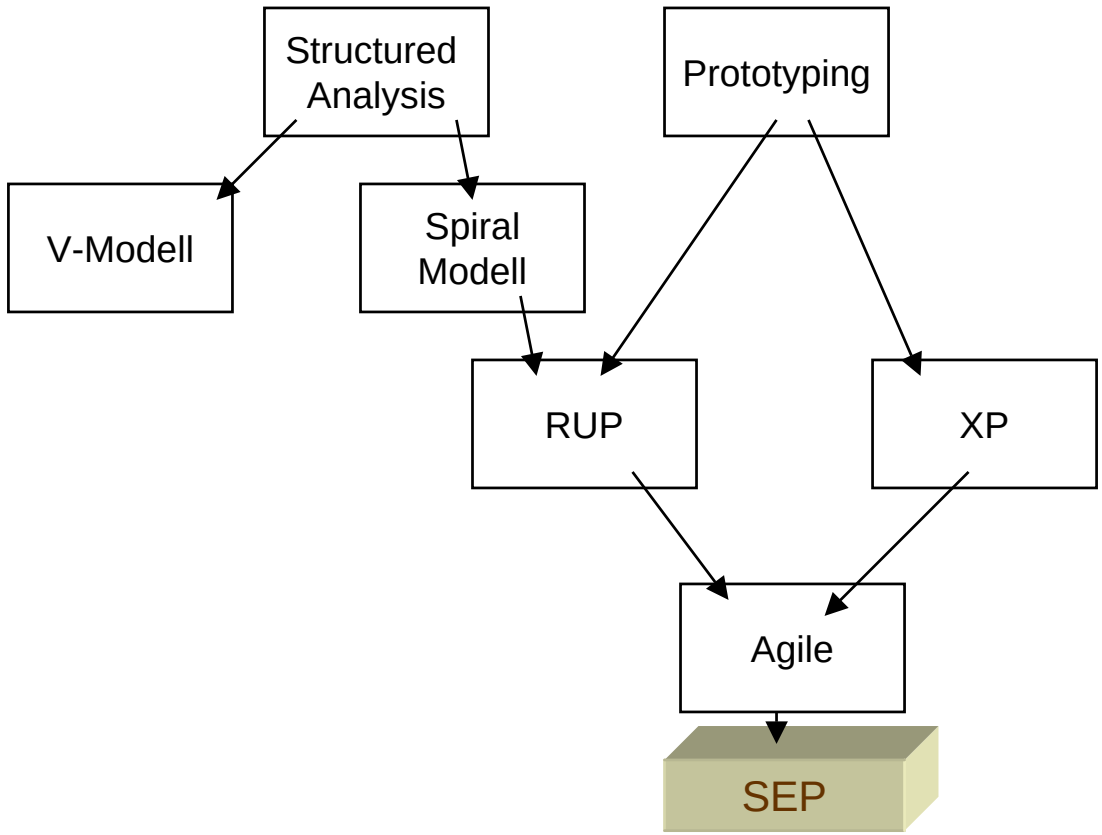
Und worauf kommt es uns nun wirklich an?

- Use cases – helfen der Verständigung mit dem Fachbereich und erzeugen das notwendige fachliche Verständnis für die Programmierer
- Kleine Inkremente – erhöhen die Planungssicherheit und schaffen Möglichkeiten zu einer adaptiven Vorgehensweise
- Code-Unit-Tests sind das Um und Auf, um feststellen zu können, ob Inkremente wirklich Inkremente sind, d.h. nachweisbare Resultate aufweisen.

- Merke! Wenn 6 Programmierer zu je 90% fertig sind, so ist das gesamte Projekt zu 48% fertig.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Fertig ...



© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004

Quellen

[Beck1999] Kent Beck, eXtreme Programming Explained – Embrace Change, Addison Wesley 1999.

[Fowl1999] Martin Fowler, Refactoring, Improving the Design of Existing Code, Addison Wesley 1999.

[High2000] James A. Highsmith III, Adaptive Software Development, A Collaborative Approach to Managing Complex Systems, Dorset House 2000

[Cobu2002] Alistair Cockburn, Agile Software Development, Addison Wesley 2002.

© 2004 Hans Hartmann, Wolfgang Keller, all rights reserved, Universität Leipzig, SS2004