

Prozessmodelle

Vorlesung: Software-Engineering für große Informationssysteme

TU-Wien, Sommersemester 2003

Bernhard Anzeletti

Überblick

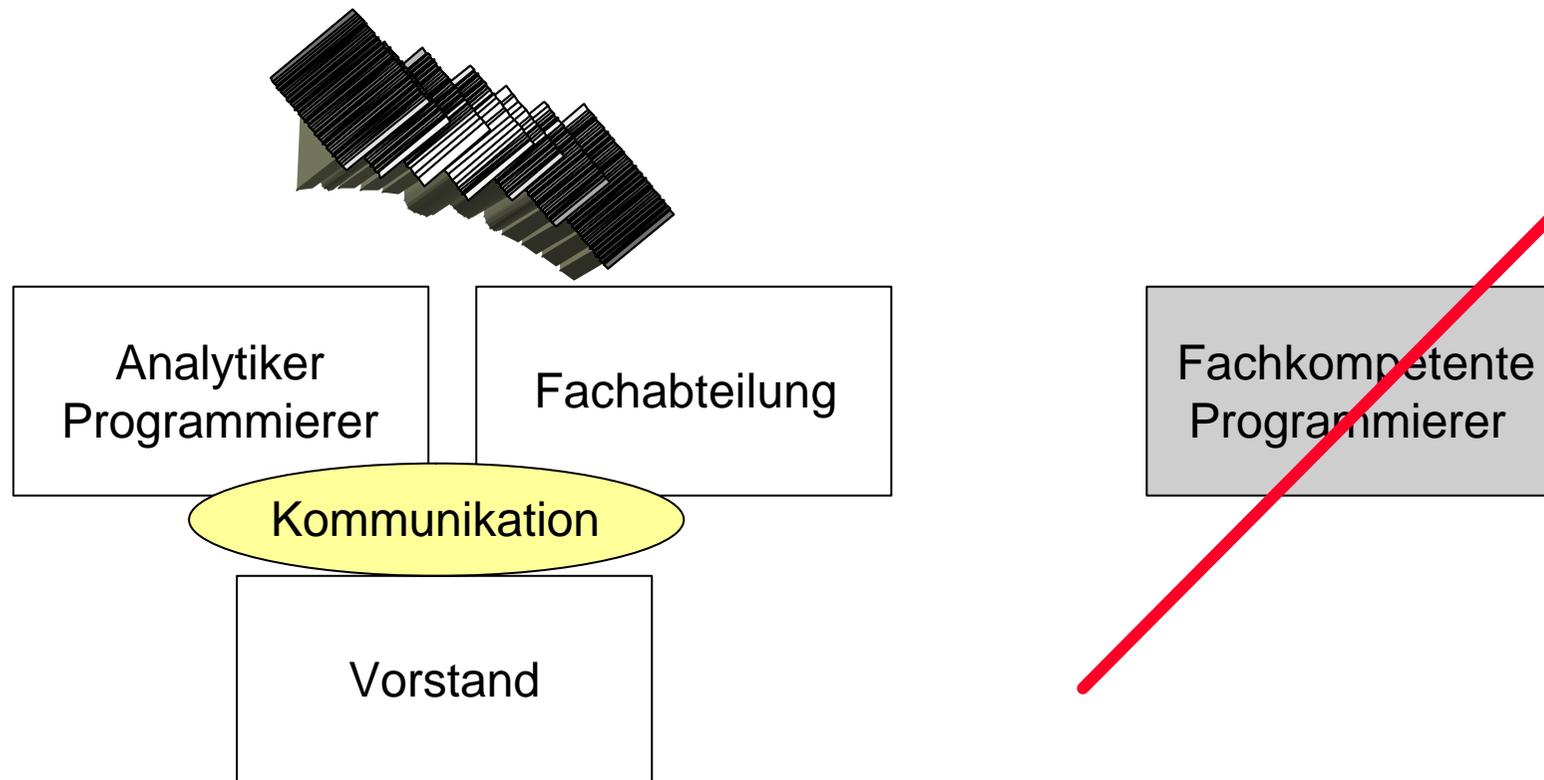
- Einführung und Forces
- Wasserfallmodell
- Prototyping
- Spiralmodell, inkrementelles Vorgehen
- RUP - Rational Unified Process
- eXtreme Programming und andere agile Prozessmodelle

Ideal...

Schnell

Fachkompetenter
Programmierer
als Anwender

... Realität



„Separation of Concerns“

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

- ...

Das Problem: Worüber reden?

Sie brauchen....

Rollenname

... Aus der Linie im Orgchart

- Jemand, der einen Nutzen erwartet und Ihr Projekt finanziert

- Jemand, der plant, nachläuft, mahnt, kontrolliert, dafür sorgt, daß was geschieht

- Jemand, der die fachliche Lösung konzipiert

- Jemand, der die technische Lösung konzipiert

- Jemand, der die Lösung programmiert

- Jemand, der die Lösung testet

- ...

Sponsor
Auftraggeber

Projektleiter

Analytiker
Key-User

Architekt

Entwickler

Tester

Vorstand

Fachbereich
EDV
(beide)

Fachbereich
EDV-Beratung

EDV

EDV

Fachbereich



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-VL)

Analytiker
Key-User

Fachbereich
EDV-Beratung

Architektur, Design

Architekt

EDV

Code

Entwickler

EDV

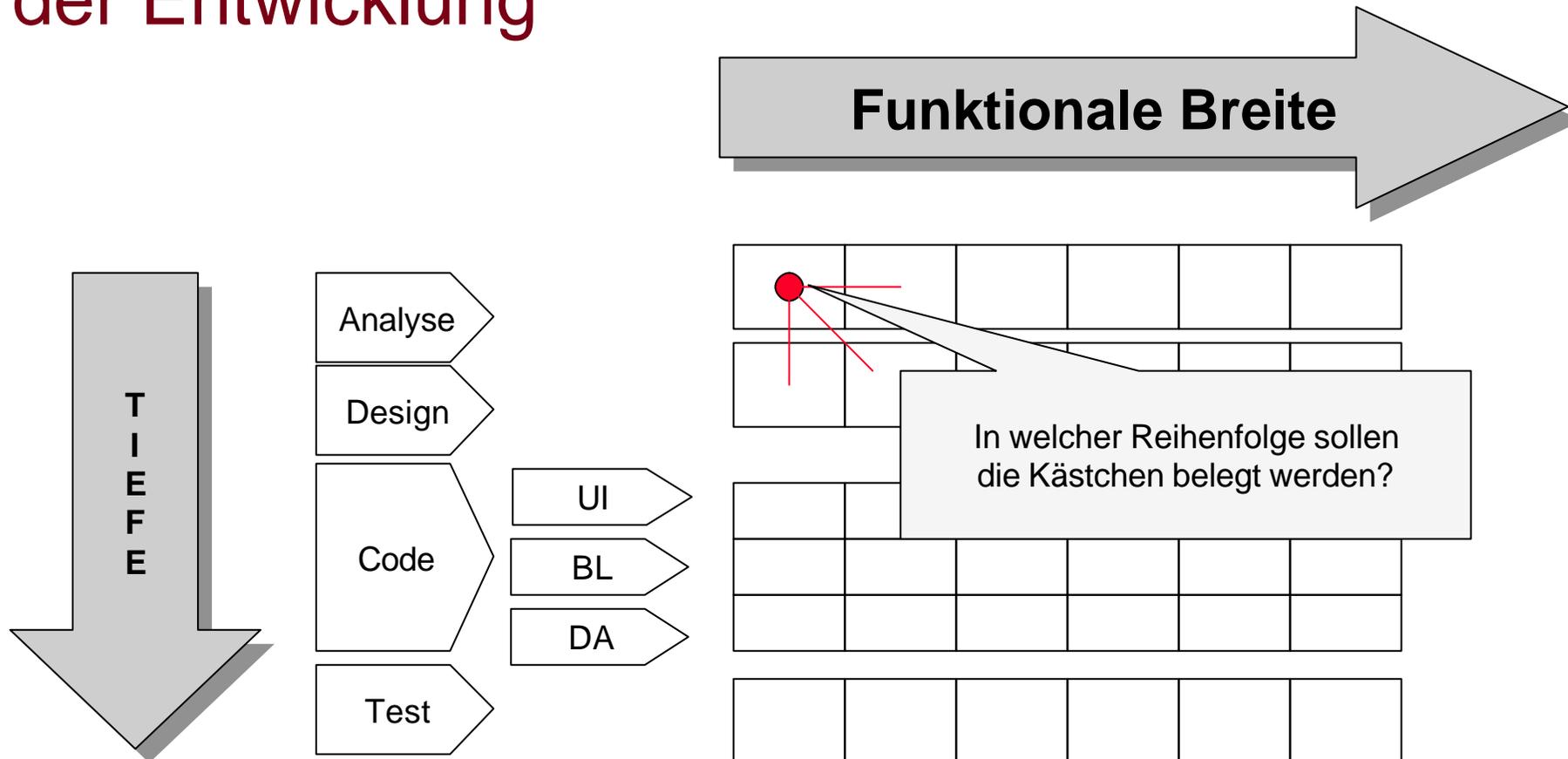
Testfälle

Tester

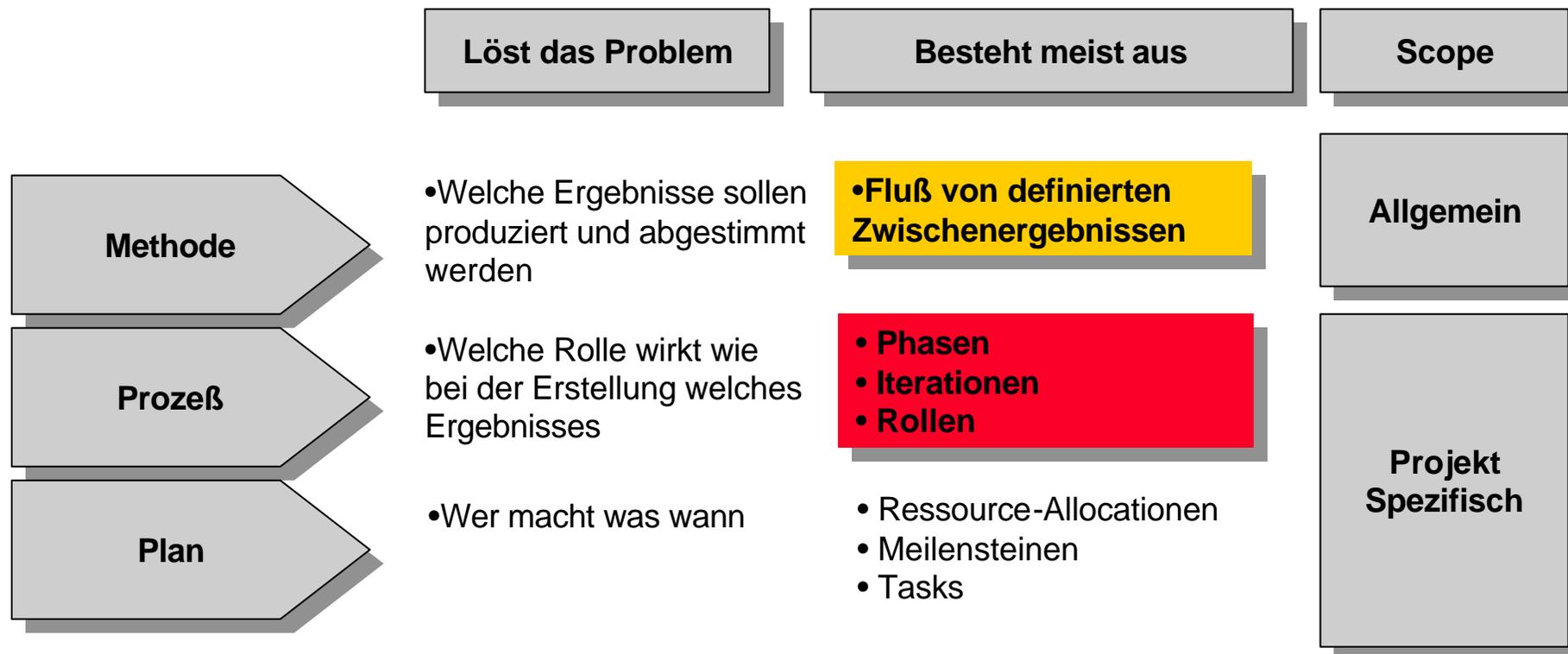
Fachbereich

...

Prozesse beschreiben die Reihenfolge der Entwicklung



Weitere Begriffe im Umfeld Entwicklungsprozeß



Ein persönlicher Tipp...

Löst das Problem

Besteht meist aus

Scope

Methode

Prozeß

Plan

**„Planen Sie die
Ergebnisse,
nicht die
Arbeit“.**

Allgemein

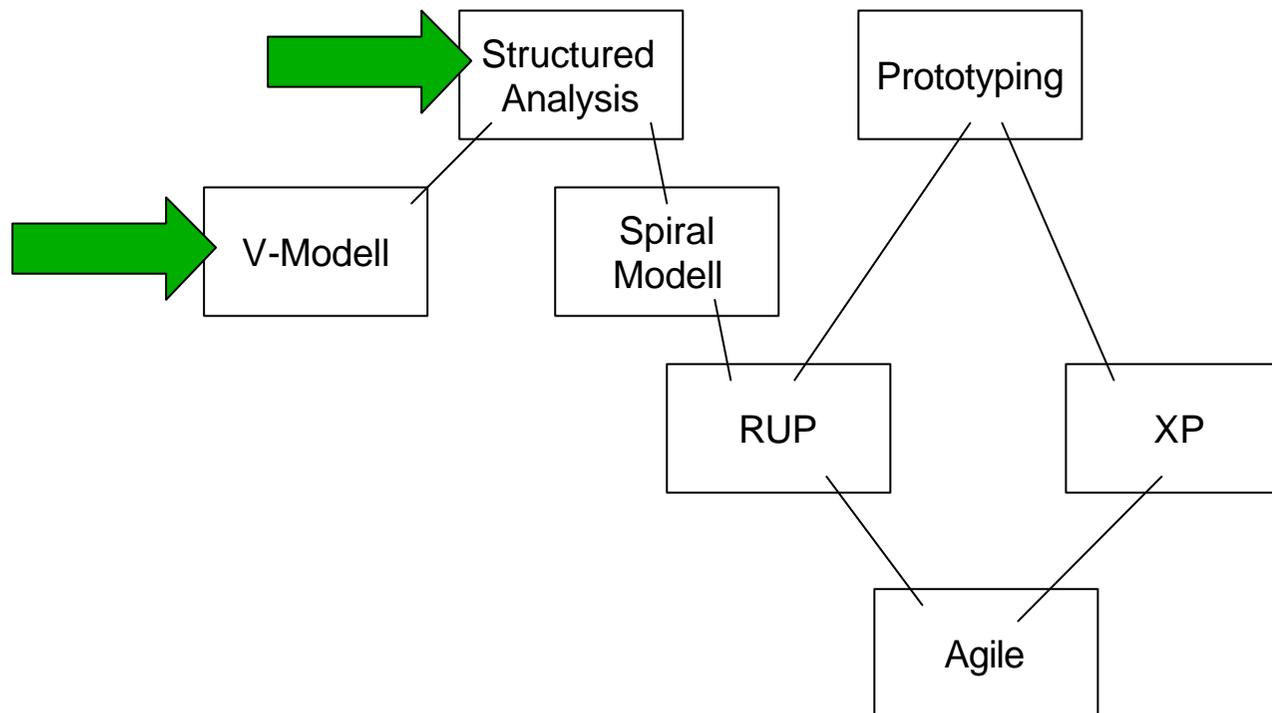
Projekt
Spezifisch

Einflußfaktoren

- 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

- Die Architektur ist daher auch Folge eines sozialen Prozesses

Fragwürdiger Stammbaum der Entwicklungsprozesse

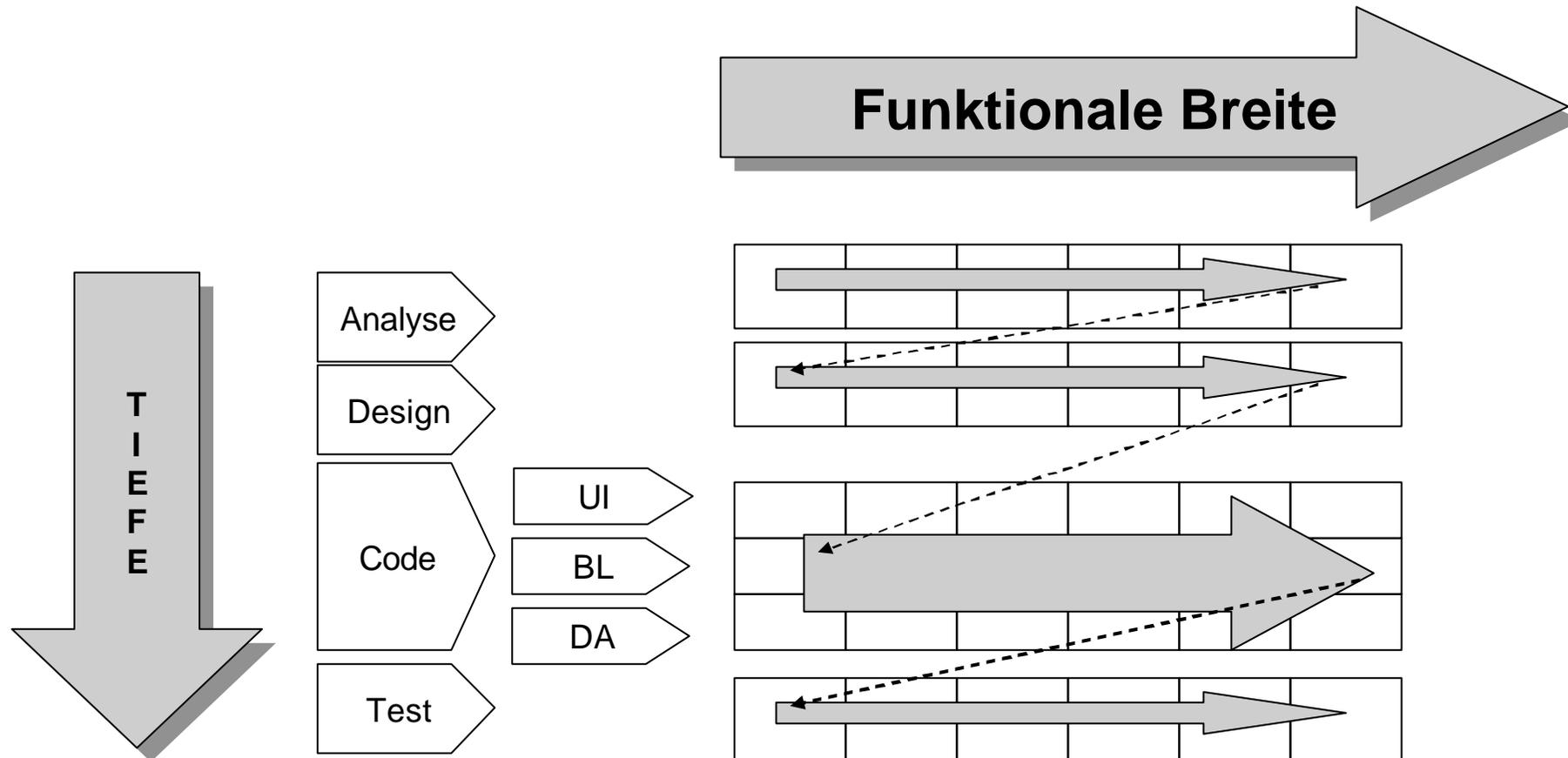


Wasserfallmodell: Structured Analysis, V-Modell

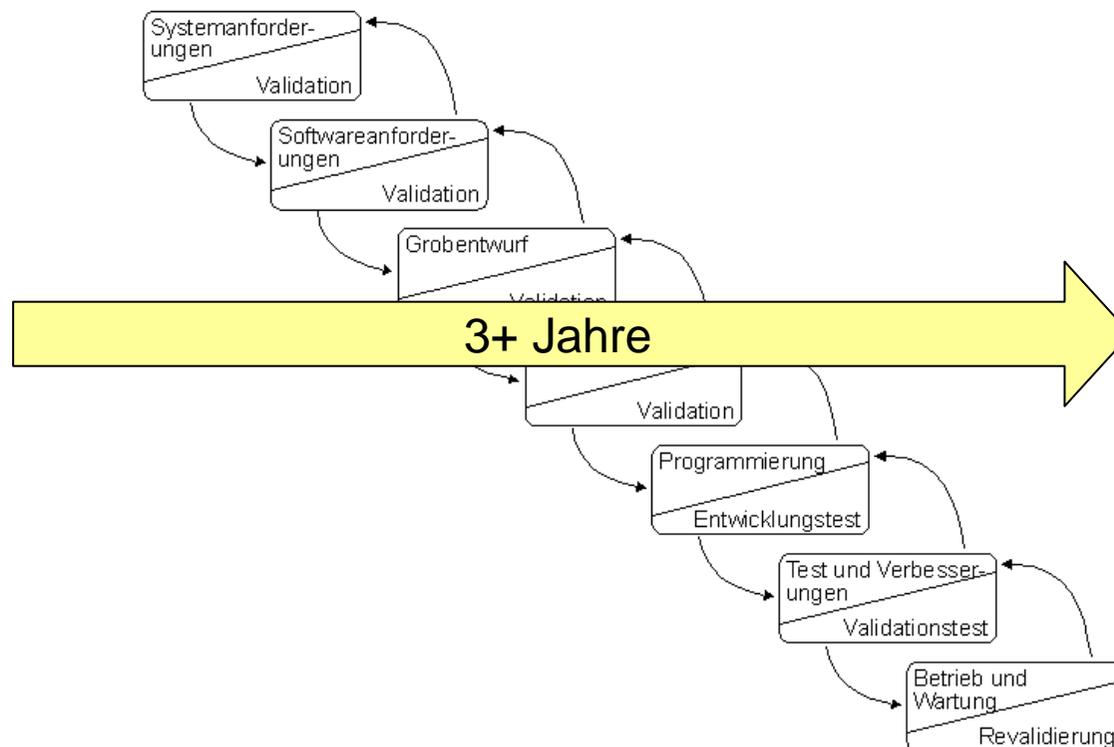
Wasserfall



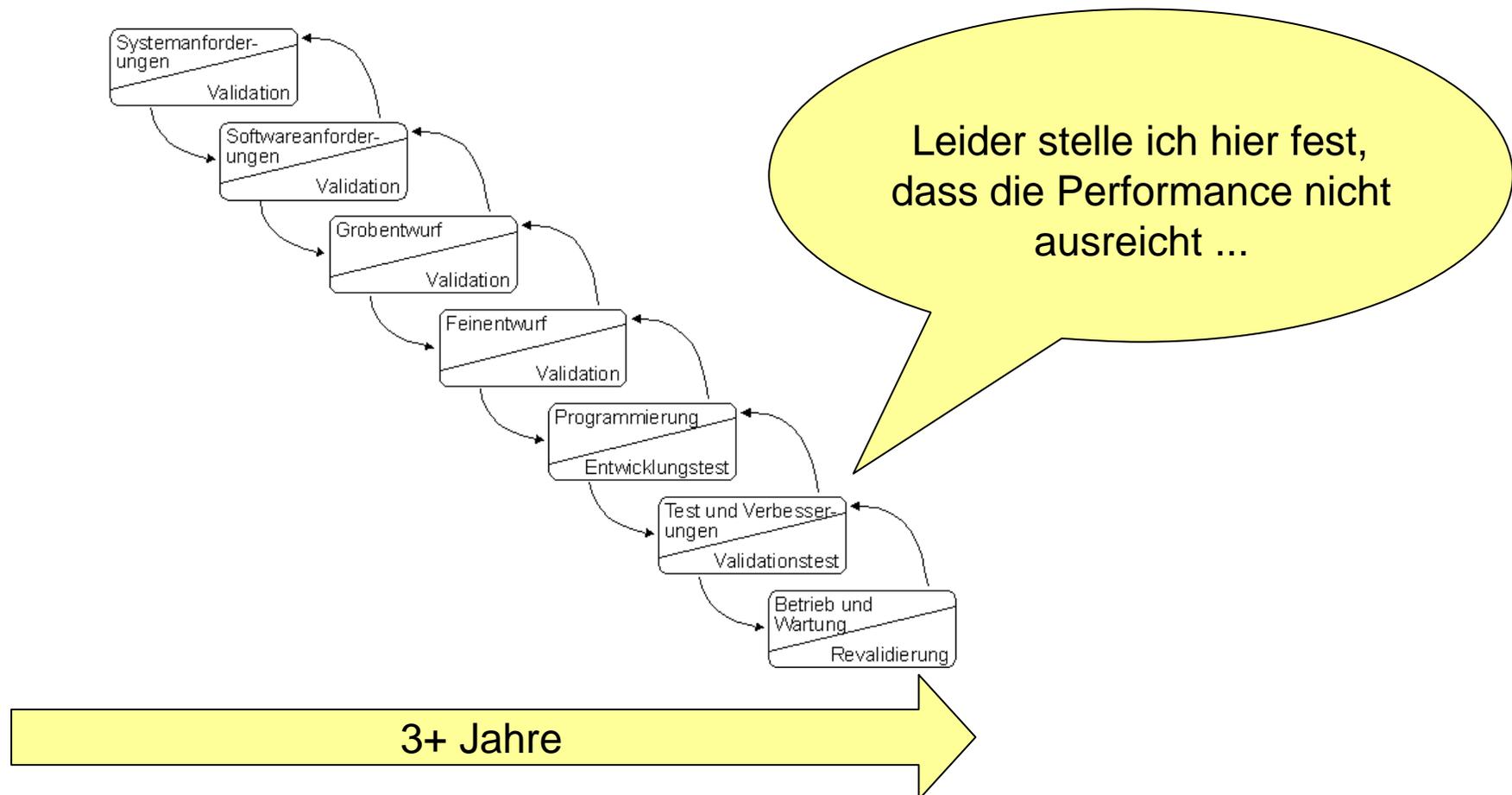
Wasserfall - Kästchenbelegungsstrategie



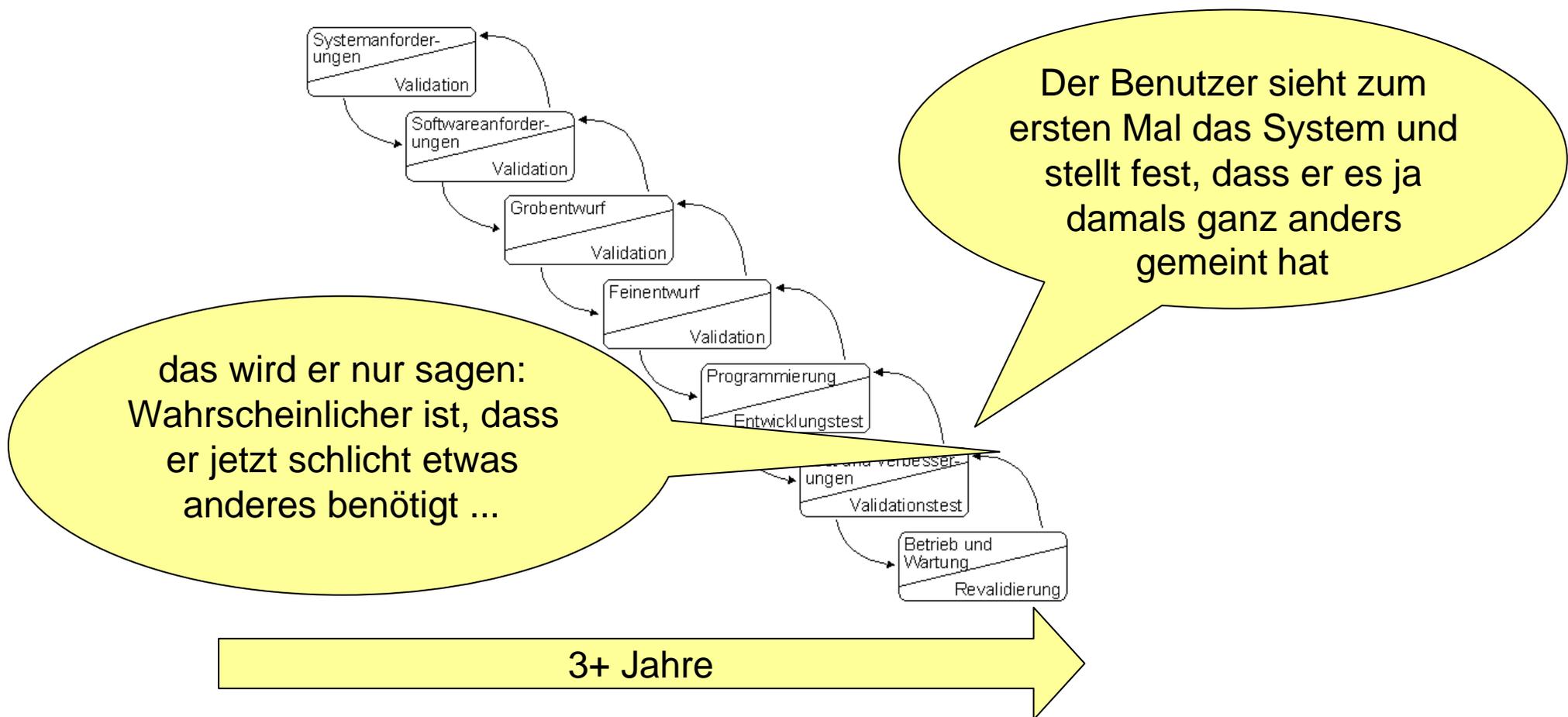
Wasserfall - eine schlechte Eigenschaft



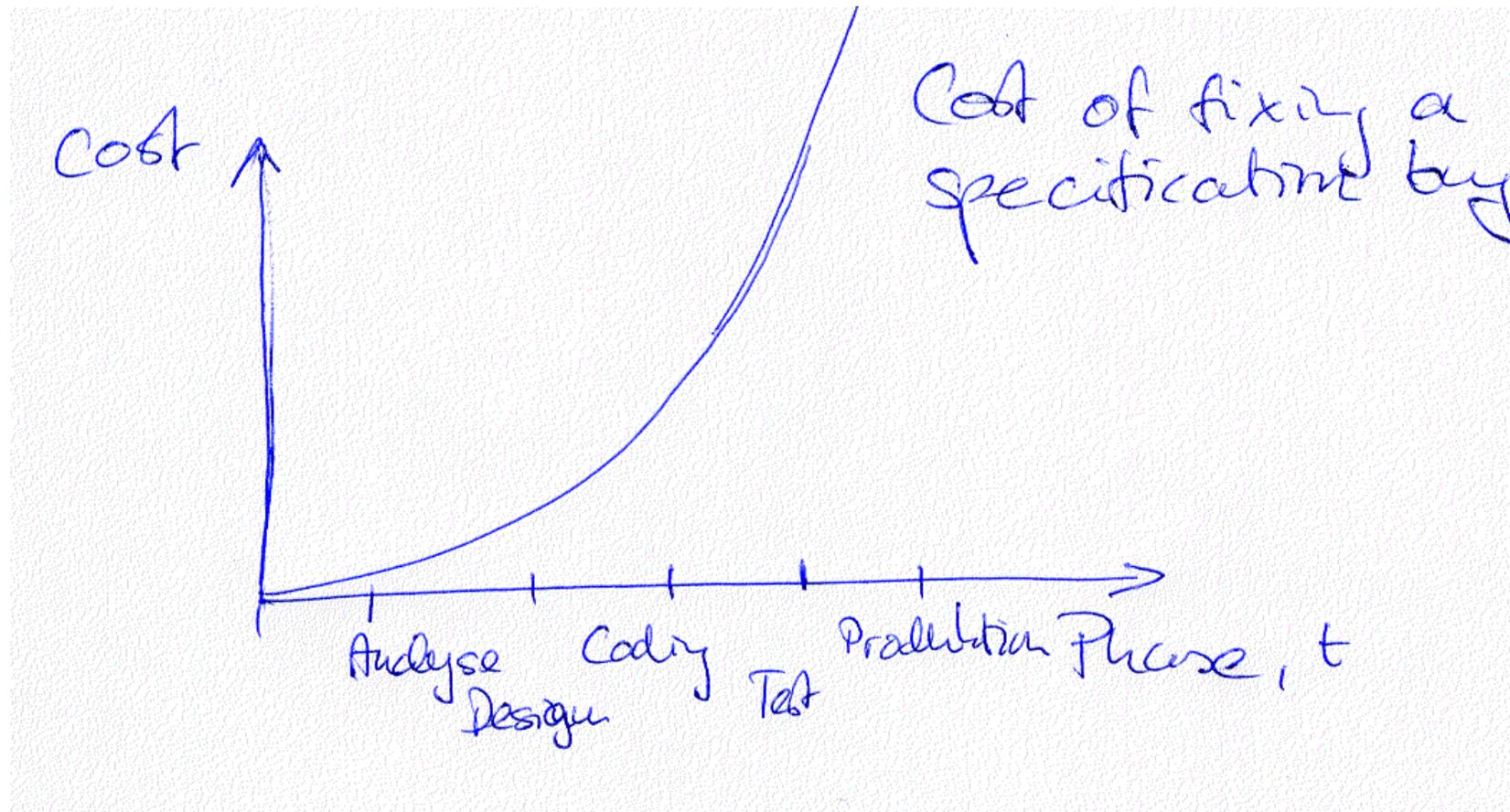
... Mit Konsequenzen ...



... Mit bitteren Konsequenzen ...



Frühere Annahme (70er) .. Fehlerkosten steigen exponentiell



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

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

Wasserfall - Nachteile

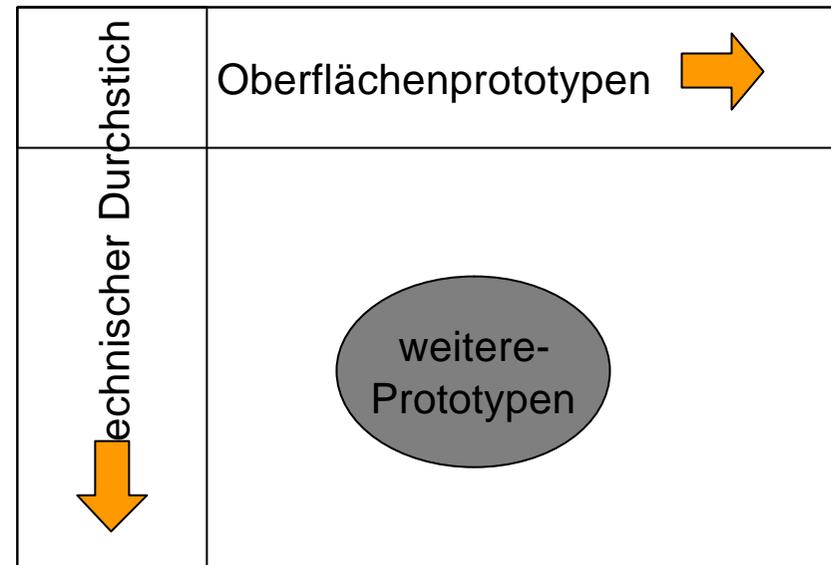
- Der Kunde sieht das Produkt sehr spät – bis dahin muss er es sich vorstellen – das überfordert 99%.
- Die meisten Kunden wollen keine Spezifikationen lesen – sie wollen Software.
- Linearer Durchlauf der Phasen ohne Änderungen ist sehr sehr unwahrscheinlich, weil sich wahrscheinlich die Vertragsgrundlage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

Wasserfall - Nachteile

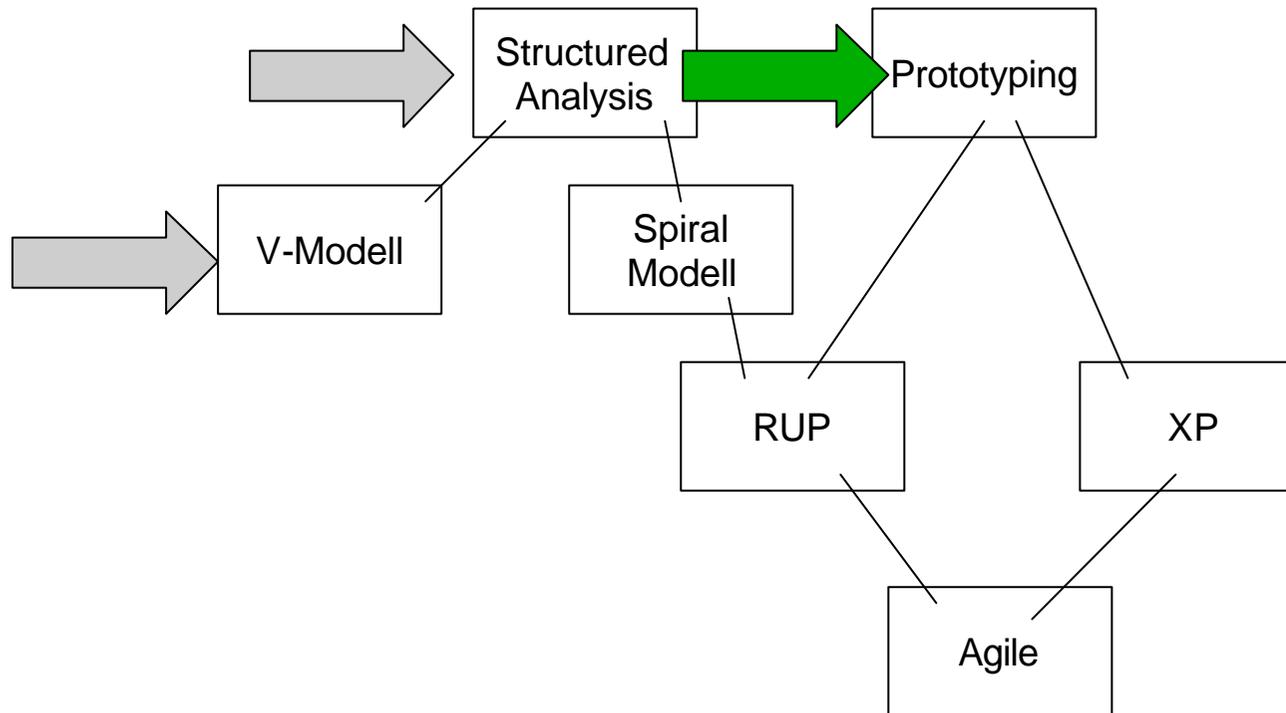
- Die Dokumente der einzelnen Phasen sind redundant. Änderungen der Geschäftsgrundlage werden über die Zeit nicht nachgezogen. Damit degeneriert die Dokumentation

Idee: Prototyping

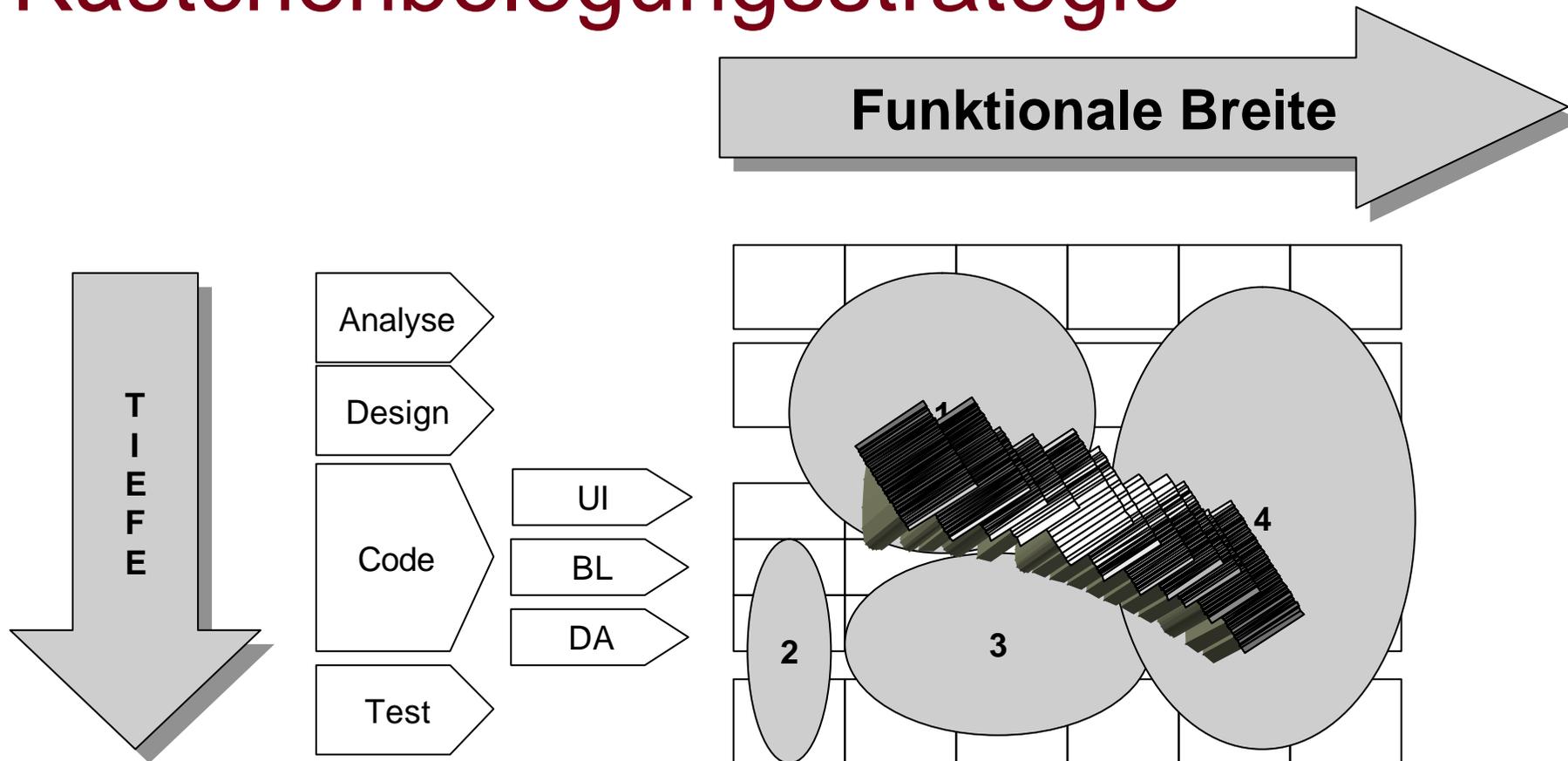
- durch technische (vertikale) Prototypen kann man vermeiden, dass man zum Beispiel die Performance Überraschung zu spät erlebt
- durch horizontale Prototypen sieht der Kunde früher was man bekommt
- komplexe Stellen sollte man mit weiteren Prototypen früh angehen („hardest things first“) – man merkt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



Fragwürdiger Stammbaum der Entwicklungsprozesse



Prototyping - Kästchenbelegungsstrategie



Prototyping ... Vorteile



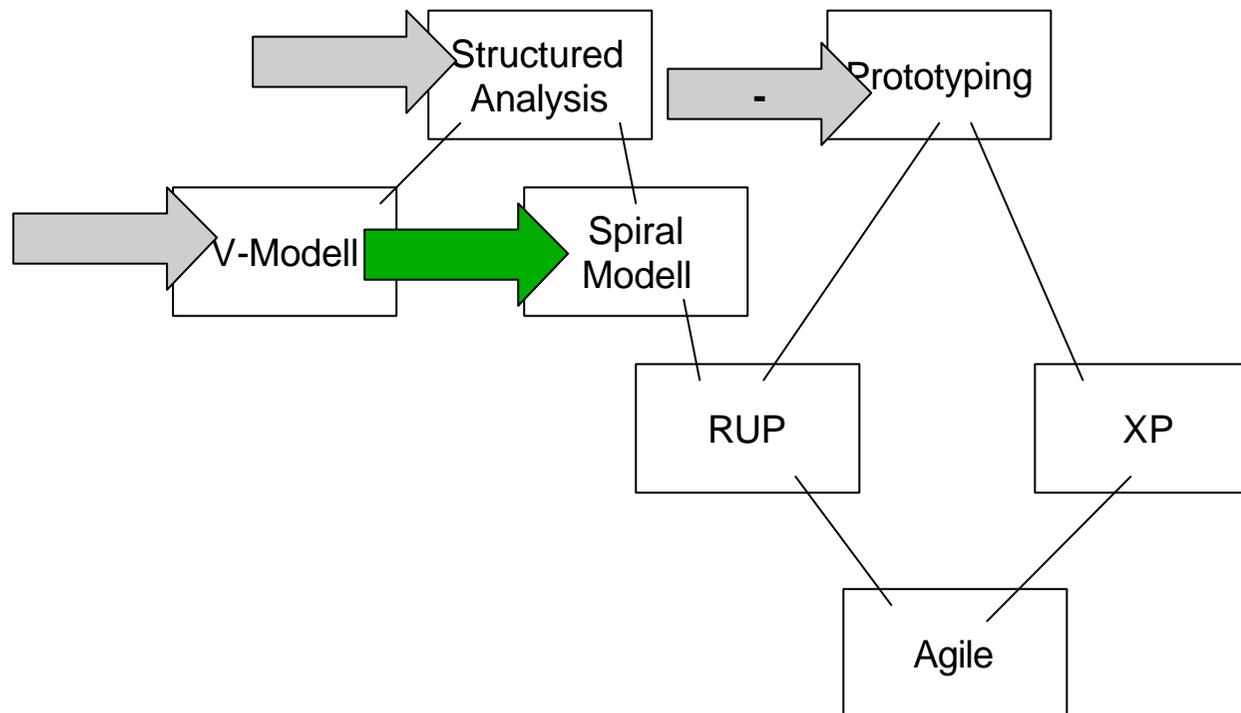
- man kann damit Risiken minimieren und frühzeitig Feedback vom Endbenutzer bekommen
- Prototypen lassen sich gut in Prozessmodelle integrieren
- Oberflächen Prototypen lassen sich mit RAD Werkzeugen schnell bauen
- Sie liefern wichtige Information über das Gesamtsystem (auch Grundlagen für FP Schätzungen)

Prototyping ... Nachteile



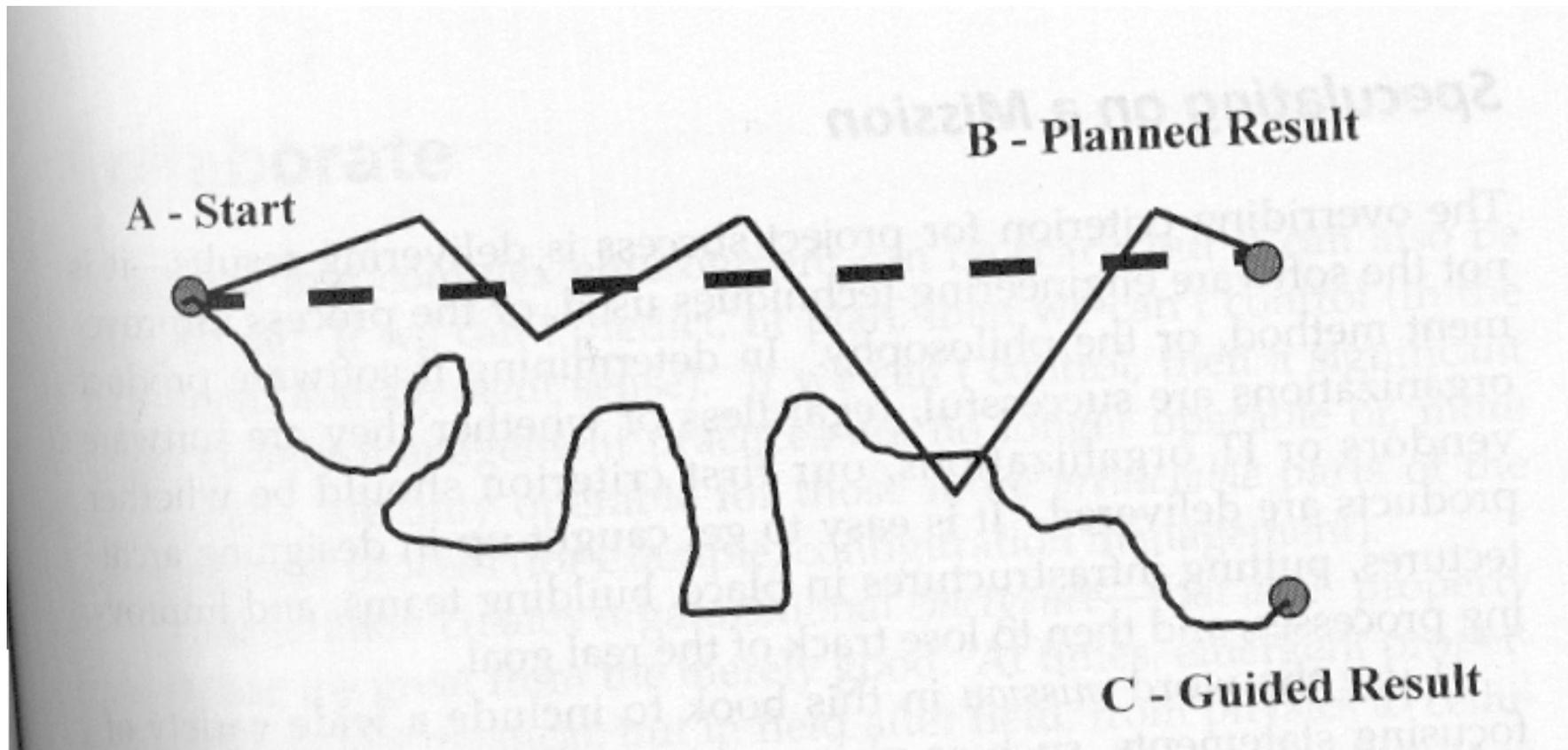
- Der Entwicklungsaufwand steigt, da von den Prototypen viel weggeworfen wird
- Es soll schon vorgekommen sein, dass das Management einen Wegwerfprototypen als das fertige Produkt gesehen hat ✍
- und damit läuft man Gefahr, beim Management unter Druck zu geraten ..

Fragwürdiger Stammbaum der Entwicklungsprozesse



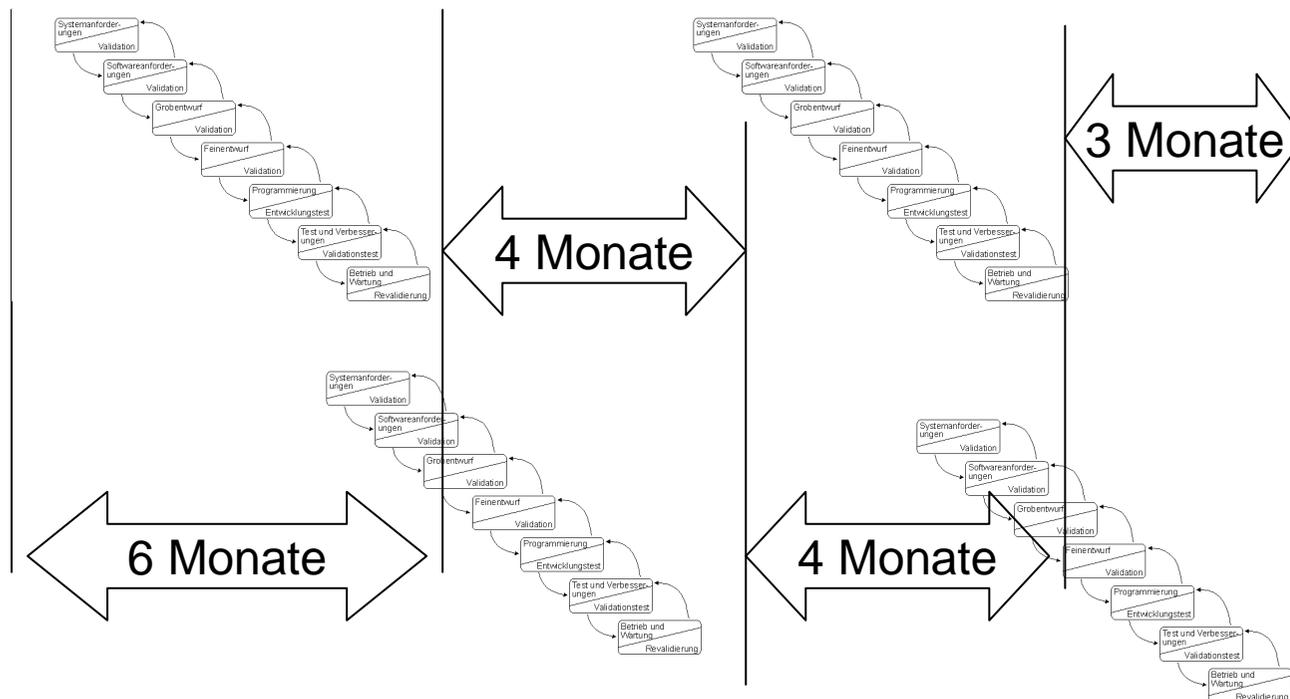
Spiralmodell

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



[High2000]

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



Überlappung ist kein Zufall ..

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

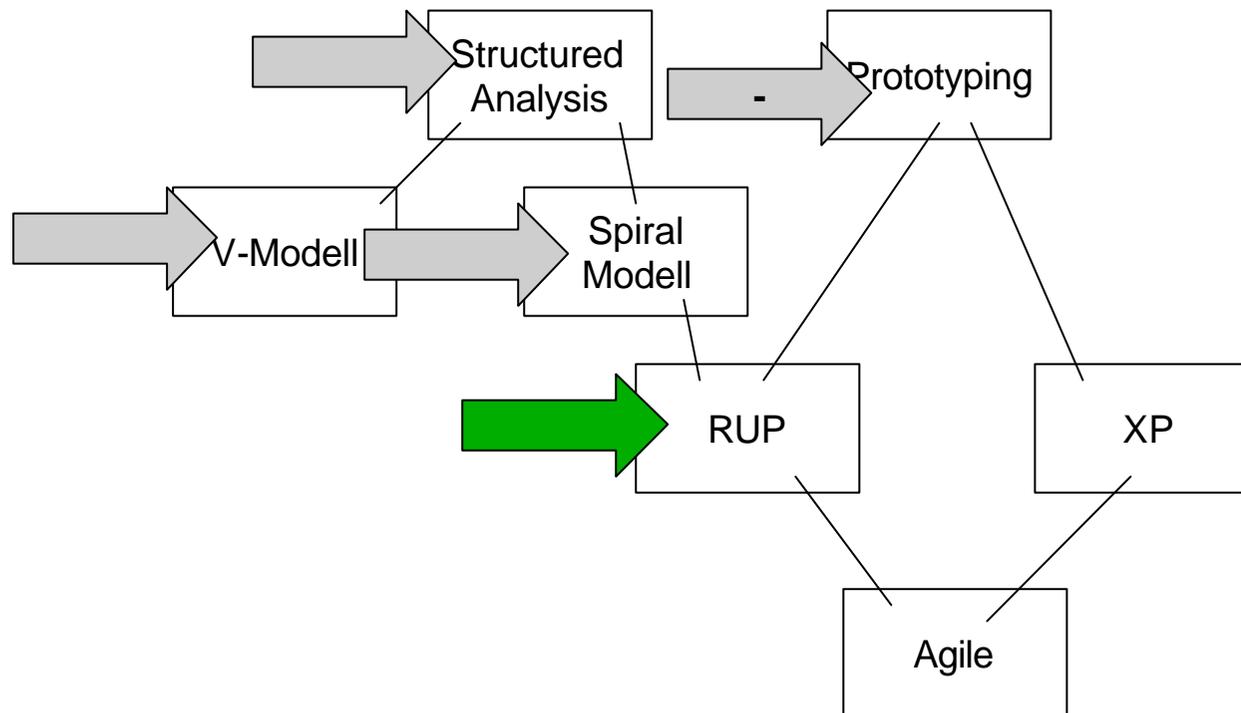
Spiralmodell Nachteile



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

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

Fragwürdiger Stammbaum der Entwicklungsprozesse



Historie RUP

OO Prozesse der „Tres Amigos“

... Und als sie miteinander sprachen...

OMT [Rumbaugh]

OOA/OOD [Coad/Yourdon]

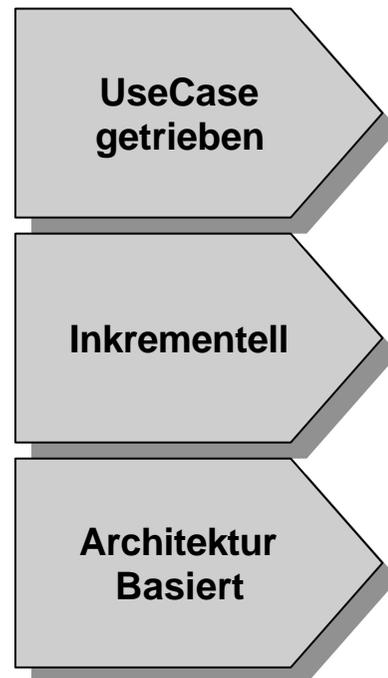
UseCaseDriven [Jacobson]

ObjectOriented Design [Booch]

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



RUP - Use-Case getrieben, Inkrementell, Architekturbasiert

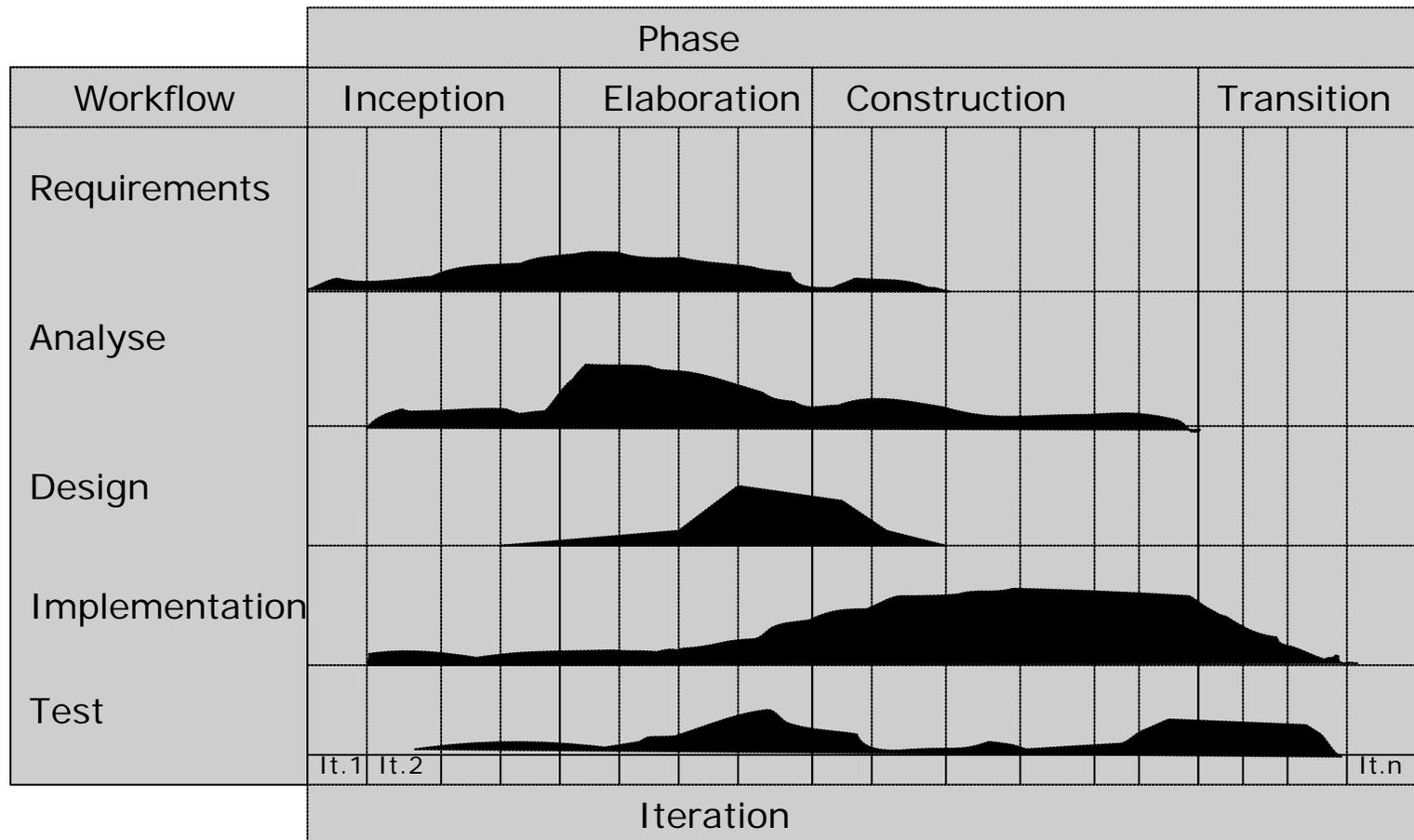


- Use Cases bilden die Basis für alle Phasen incl. Test
- Die Planung erfolgt nach UseCase-Paketen und Fertigungstiefe

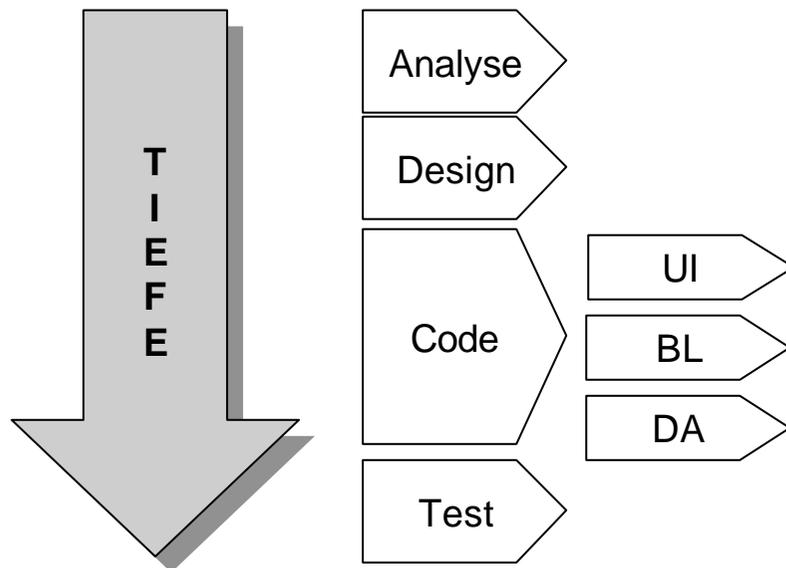
- Das System wird in Iterationen errichtet
- Jede Iteration kann etwas mehr als die Vorgängeriteration

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

RUP - Phasen und Workflows



RUP - Kästchenbelegungstrategie



Funktionale Breite

Lieber Projektleiter,

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

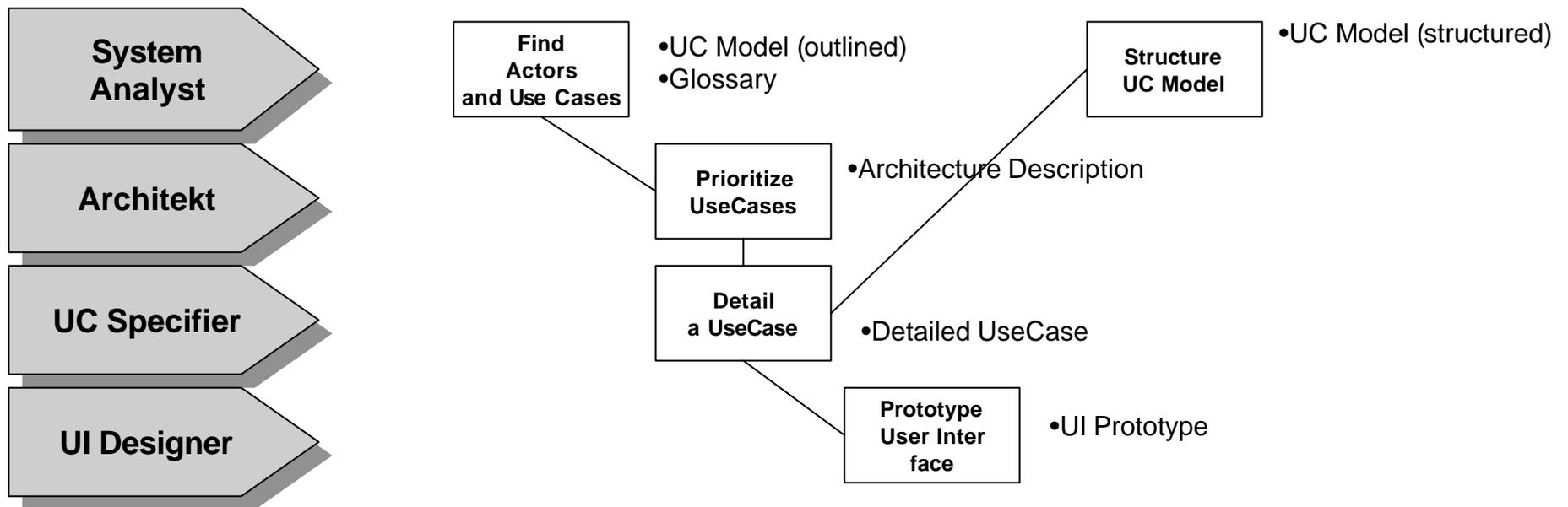
Wenn Du aber verschiedene Schwerpunkte setzt - Risikominimierung, Analyse, Architektur, Code, Test, etc - helfen wir Dir gerne:

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

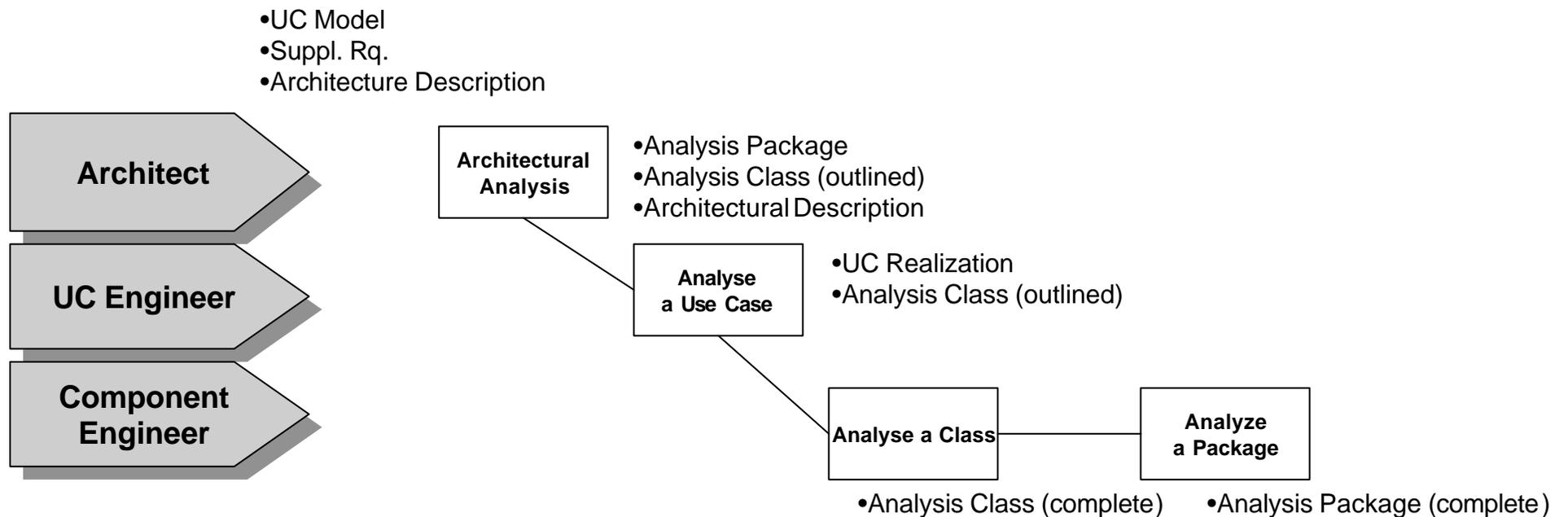
Die Phasenbeschreibung sagt Dir, wie Du planst (worauf Du achten mußt).

RUP -Requirements WF

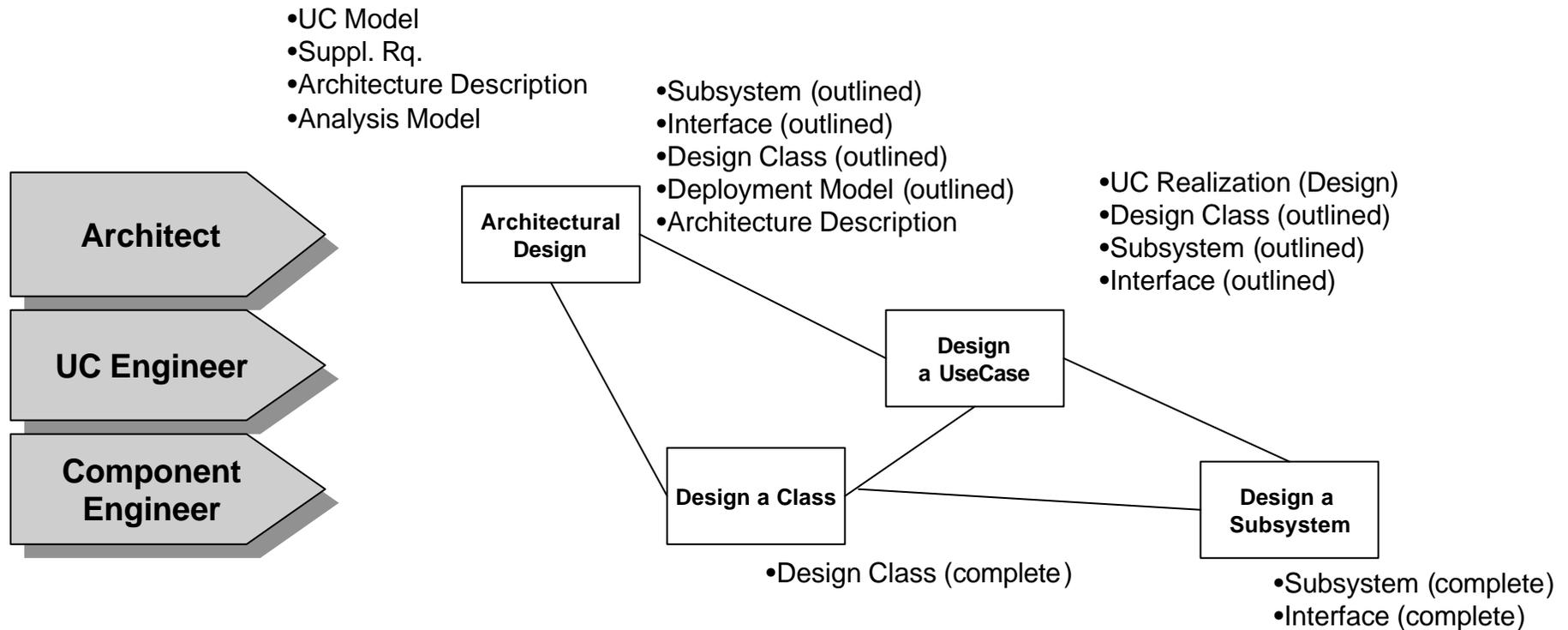
- Domain Model
- Feature List
- Supplementary Rq.



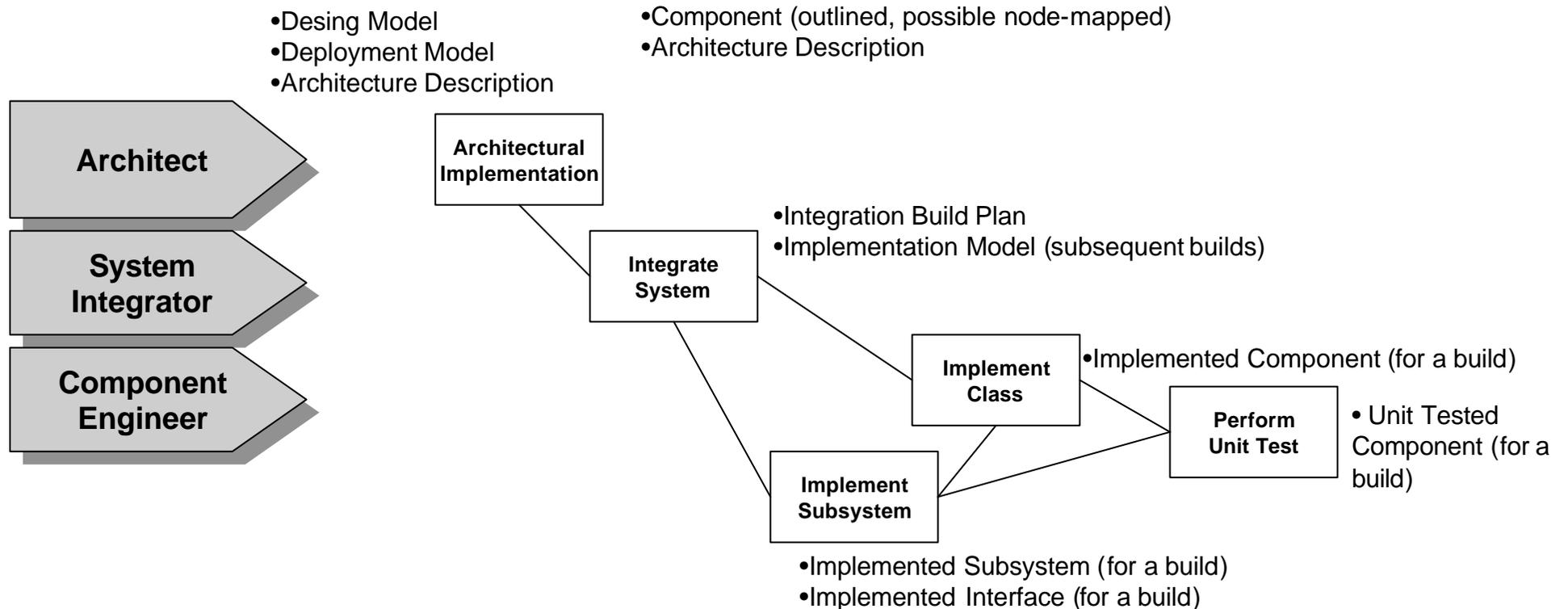
RUP -Analysis WF



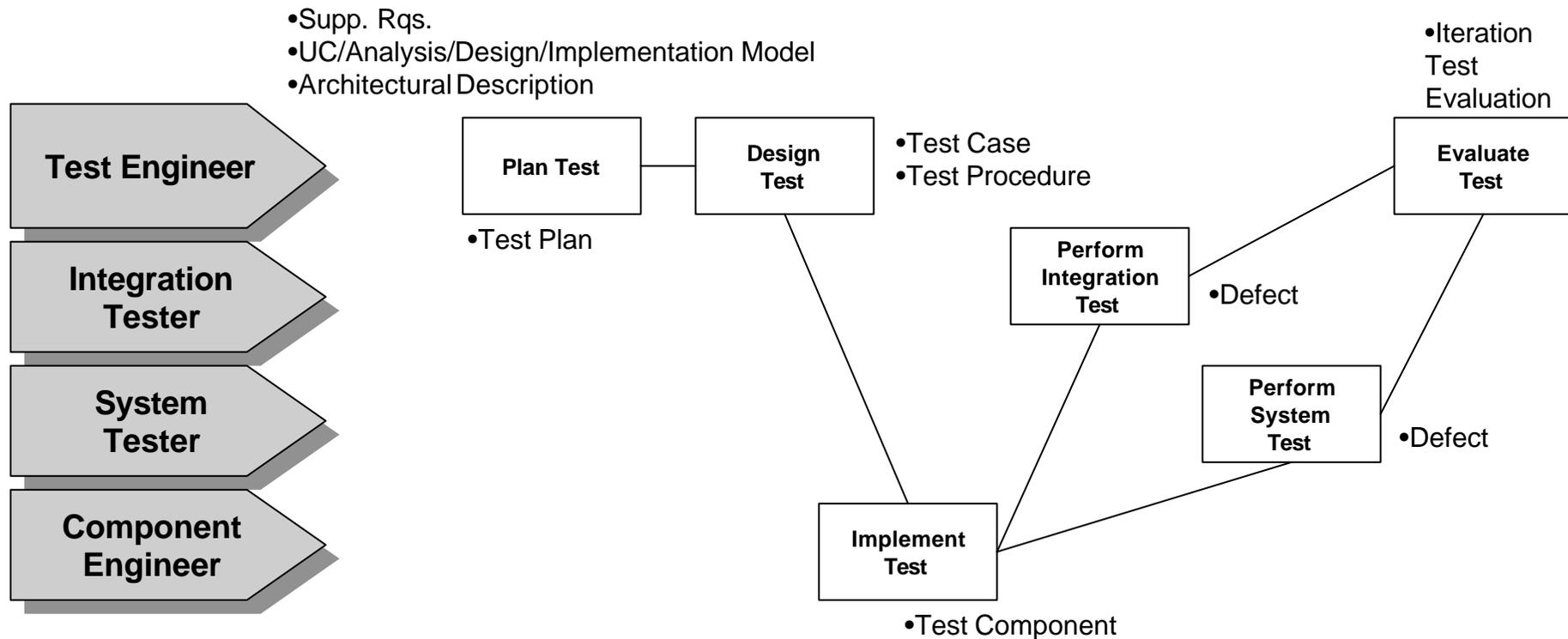
RUP -Design WF



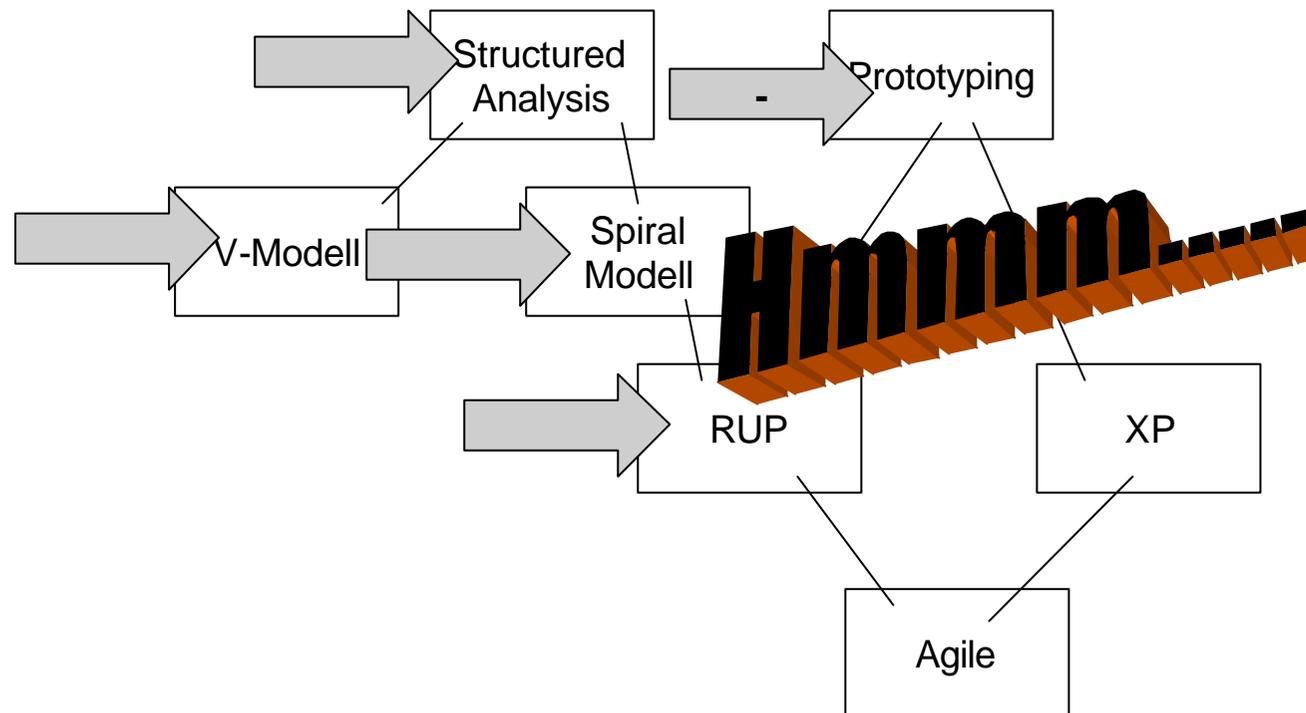
RUP - Implementation WF



RUP - Test WF



Fragwürdiger Stammbaum der Entwicklungsprozesse



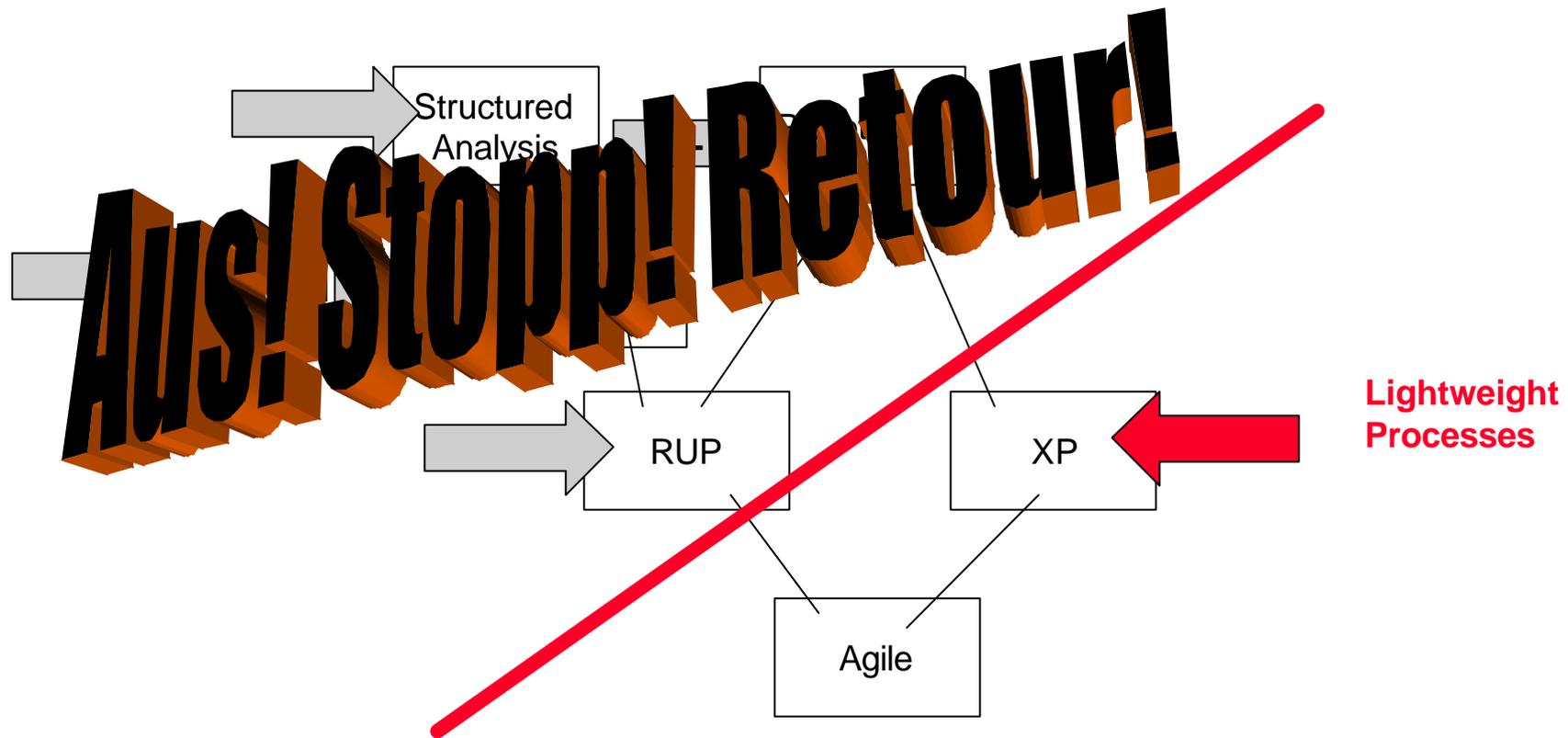
Grundprinzip traditioneller Entwicklungsmethoden



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

- **Das Individuum zählt nicht!**
- **= MACHT KEINEN SPASS**

Fragwürdiger Stammbaum der Entwicklungsprozesse



eXtreme Programming als eine mögliche agile Methode

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 ...

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**
- Integrationstests sind wichtig
 - deshalb wird permanent integriert und wieder getestet. **Continuous Integration**

Die einzelnen Praktiken von XP sind nicht neu ..



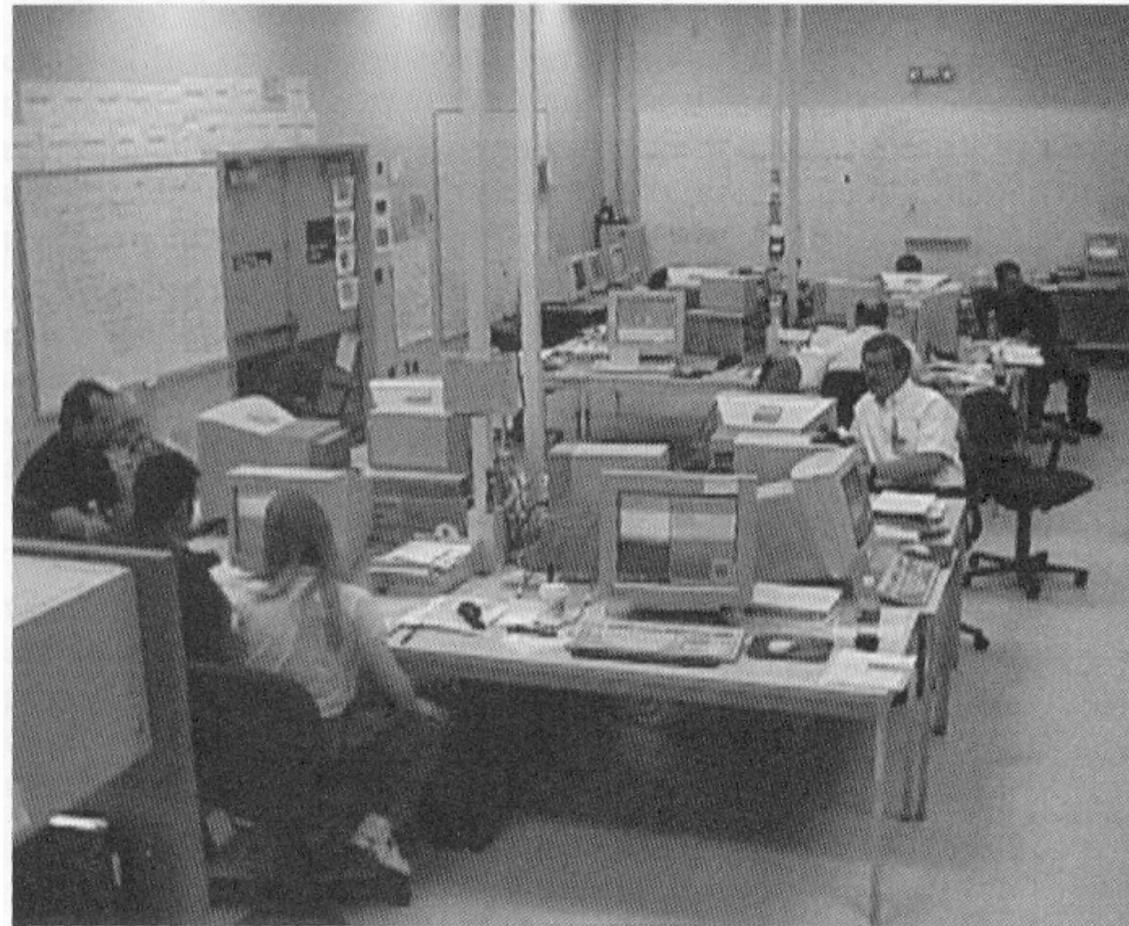
- Design ist wichtig
 - deshalb wird es während des Codierens durch **Refactoring** ständig verbessert
- Einfachheit ist gut
 - als wir das einfachste gemacht, was in diesem Moment funktioniert
 - „**do the simplest thing that might possibly work**“
- Architektur ist wichtig
 - Also wird die Architektur permanent weiterentwickelt und es werden Metaphern verwendet

Die einzelnen Praktiken von XP sind nicht neu ..

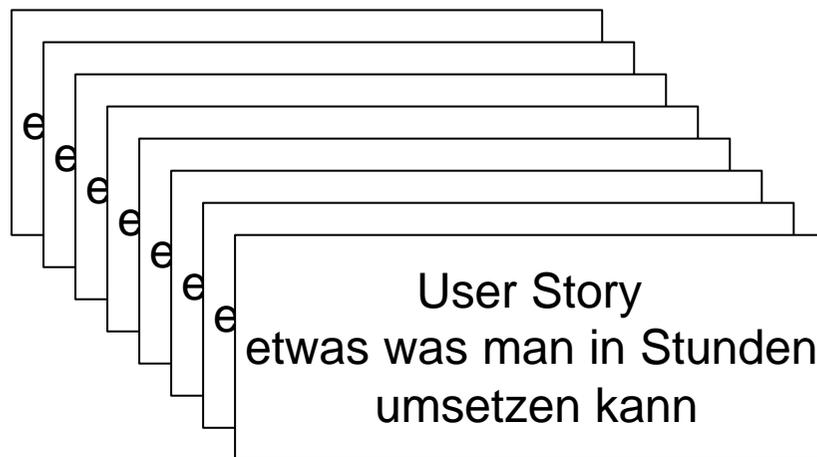


- Kurze Iterationen sind wichtig
 - also werden die Iterationen extrem kurz gemacht
 - 10 Tage, dann wieder „Planning Game“
- Feedback durch den Kunden ist wichtig
 - also ist der Kunden im selben Raum
 - **on site customer**
- **40 Stunden Woche**, weil die Leute sonst ausbrennen, wie man in der EDV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

Beispiel: Raumaufbau ...



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 ...

Beispiel für eine Story Card

Customer Story and Task Card BIW Development / COLA

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

STORY NUMBER: ~~1275~~ / 1275 PRIORITY: USER: TECH:

PRIOR REFERENCE: _____ RISK: _____ TECH ESTIMATE: _____

TASK DESCRIPTION:
 SPLIT COLA: When the COLA rate chgs. in the middle of the BIW Pay Period user 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 DE Ent Emges COLA

Date	Status	To Do	Comments	STN

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

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 ...

Was bitte ist Refactoring? Beispiel 1

204 ORGANIZING DATA

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.

```
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;
```

Replace Magic Number with Symbolic Constant

Quelle:
[Fowl1999]

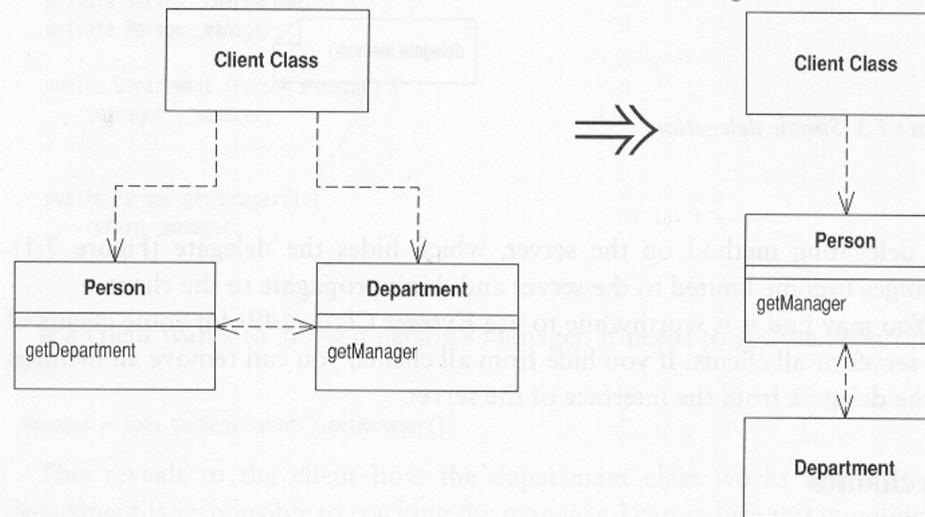
Was bitte ist Refactoring?

Beispiel 2:

Hide Delegate

A client is calling a delegate class of an object.

Create methods on the server to hide the delegate.



Quelle:
[Fowl1999]

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.

Refactoring

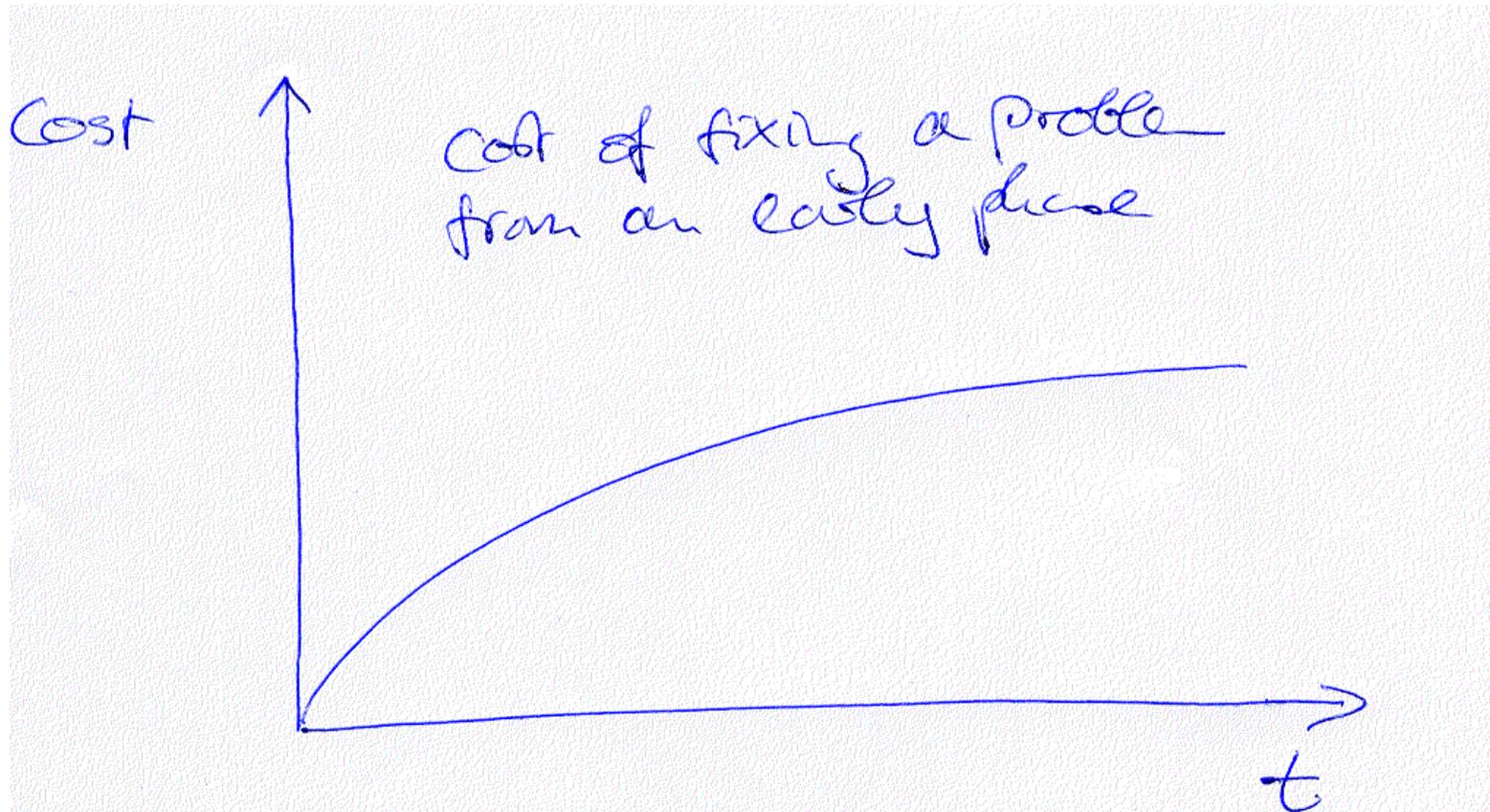
- Refactoring wurde nicht von Martin Fowler erfunden sondern von William Opdyke, der bei Ralph Johnson gearbeitet hat
- Refactorings sind Pattern (Muster) mit vorher/nachher Zuständen, wie man Code verbessern kann
- Refactoring funktioniert besser, wenn man ein Tool hat, das Refactoring unterstützt.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

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

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



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 im Chrysler C3 Project erfunden und praktiziert ...
 - keiner weiß, ob es für große Projekte skaliert
- XP ist ein Hype
 - Deshalb redet jeder gerne drüber

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 ..

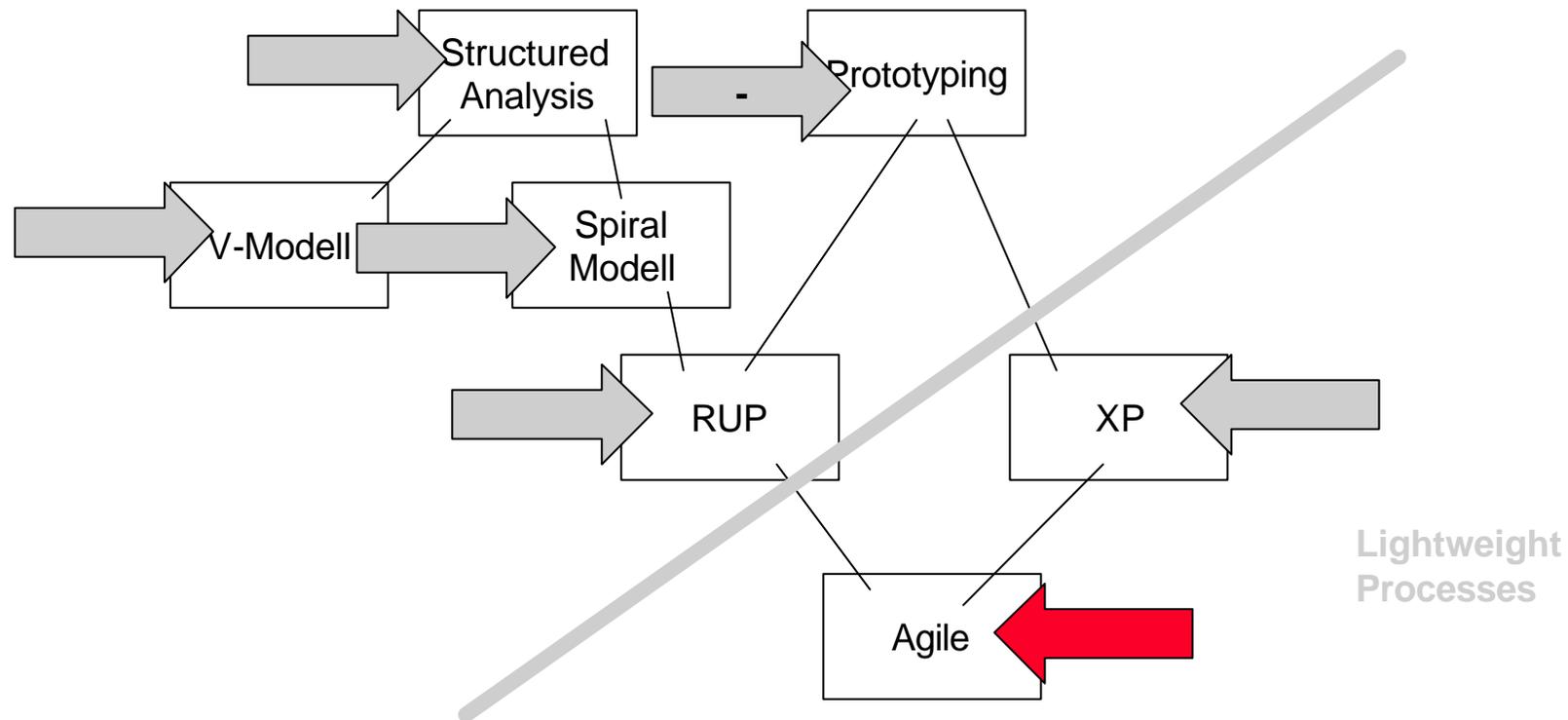
Vorsicht Hype

- 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 „politischen Gründen“

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

Fragwürdiger Stammbaum der Entwicklungsprozesse



Agile Prozesse

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

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

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]

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

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

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?

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?

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



- on site customer
 - weil viele Unternehmen keine Projektkultur haben
 - und Mitarbeiter der Fachbereiche 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“

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



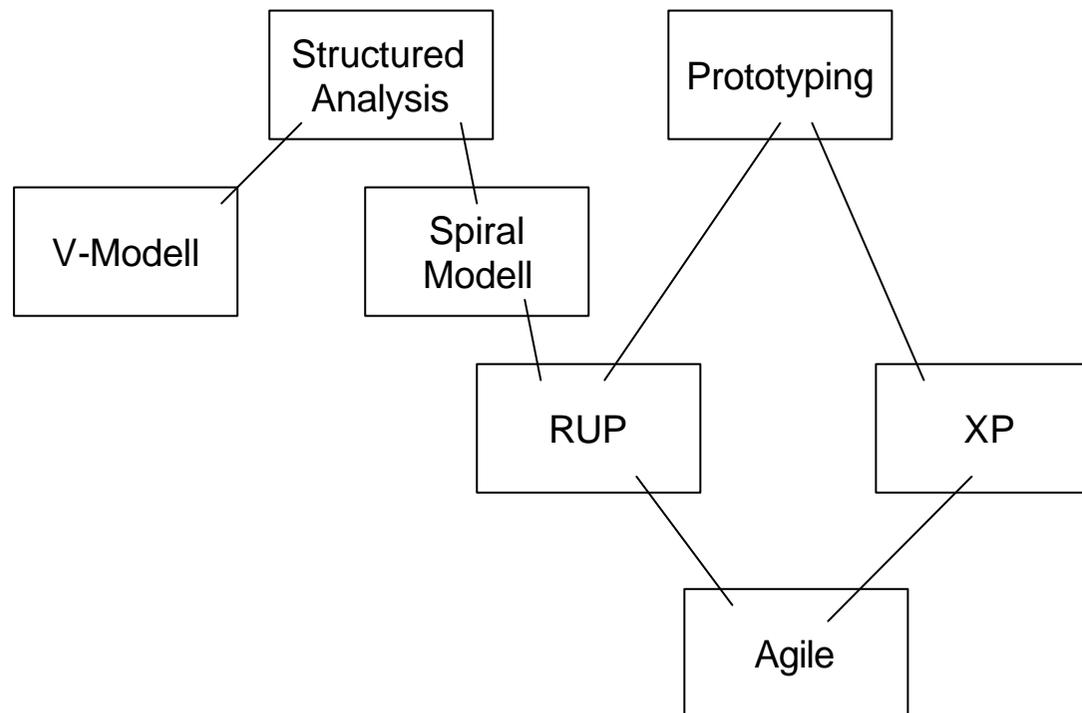
- Metaphern
 - erfordern Liebe zum Thema, großes Wissen und Liebe zum Job.
 - habe ich noch wenig gesehen

Agile Prozesse: Was von XP haben wir noch wenig probiert



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

... Fertig!



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.