

Organisation und Projektmanagement der IV

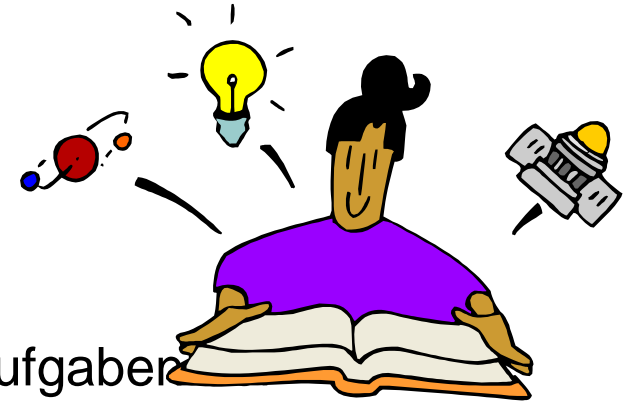
03 Vorgehensmodelle; Stand: 2002-10-28

Prof. Dr. Dirk Stelzer



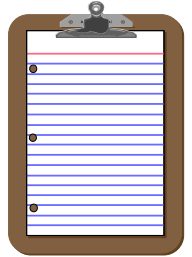
Technische Universität Ilmenau
Fakultät für Wirtschaftswissenschaften
Fachgebiet Informationsmanagement

Lernziele



- Sie kennen wesentliche (fachlich-technische) Teilaufgaben Systementwicklung.
- Sie kennen die Bedeutung von Vorgehensmodellen für die Entwicklung betrieblicher Anwendungssysteme.
- Sie kennen die Eigenschaften wichtiger Vorgehensmodelle.
- Sie können erörtern, welche Vorgehensmodelle sich unter bestimmten Rahmenbedingungen mehr oder weniger eignen.

Gliederung



- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Extreme Programming
- Weitere Modelle
- Empirische Befunde

Unkoordinierte Entwicklung

„The hacker or code-and-fix model“:

- Write some code
- Fix the problems in the code

Probleme / Gefahren

- Entwicklungserfolg fast ausschließlich vom Können der Entwickler abhängig
- Unstrukturierter Code
- Projektfortschrittskontrolle schwierig
- Oft werden Kundenanforderungen nicht erfüllt
- Koordination großer Entwicklerteams schwierig / unmöglich
- ...



Es gibt eine Vielzahl von Vorgehensmodellen ...

... Phasenkonzepten, Lebenszyklusmodellen etc.

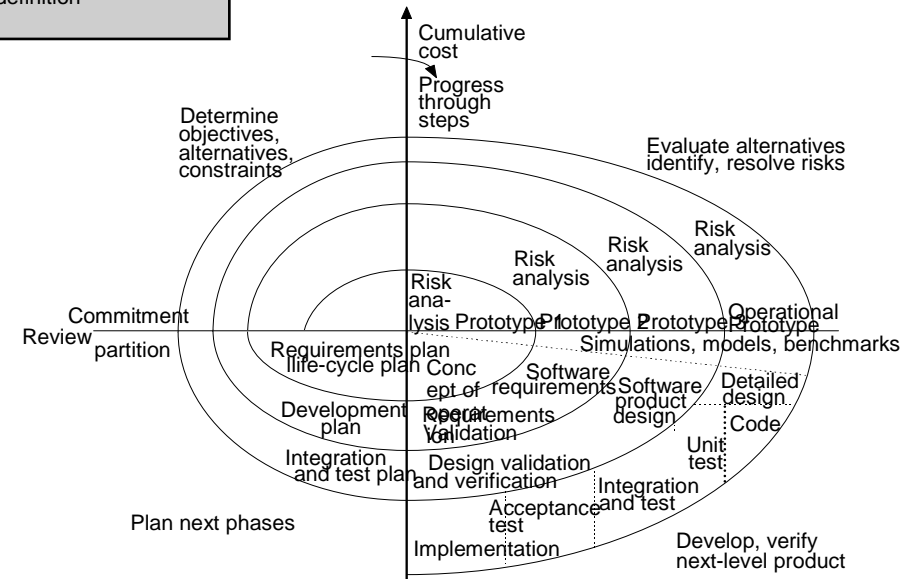
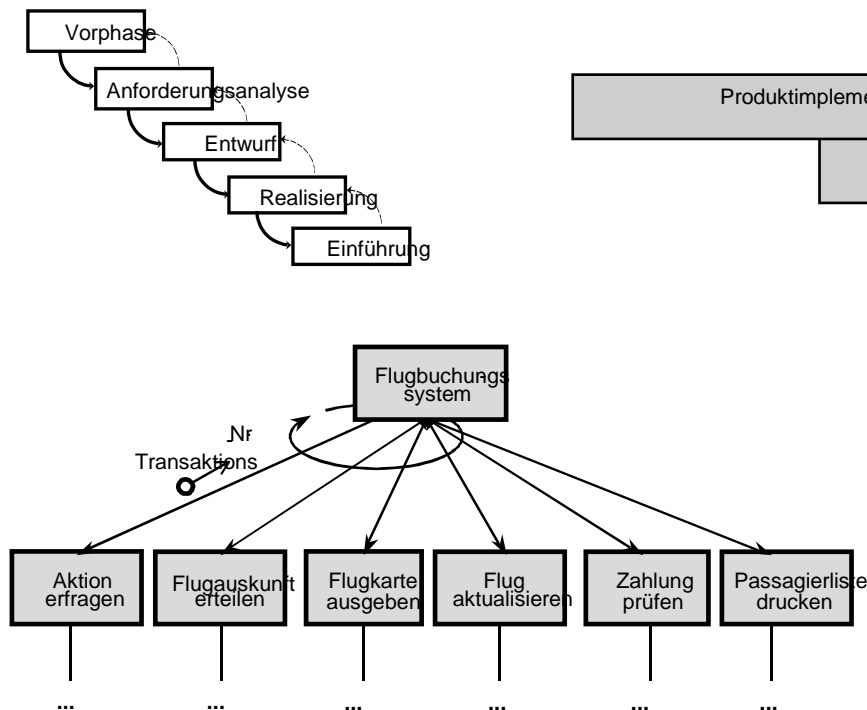
Sie unterscheiden sich u.a. hinsichtlich ...

- der zeitlichen Abfolge der Bearbeitung von Aufgaben,
- der Annahmen über die Form der Arbeitsteilung, der Abstimmung von Teilaufgaben und der Aufbauorganisation (der Entwicklungseinheit) sowie
- der Empfehlungen zu Dokumentation, Erzeugung von Zwischenprodukten und bestimmter Hilfsmittel (z. B. Prototyping)
- ...



Vorgehensmodelle für die Softwareentwicklung

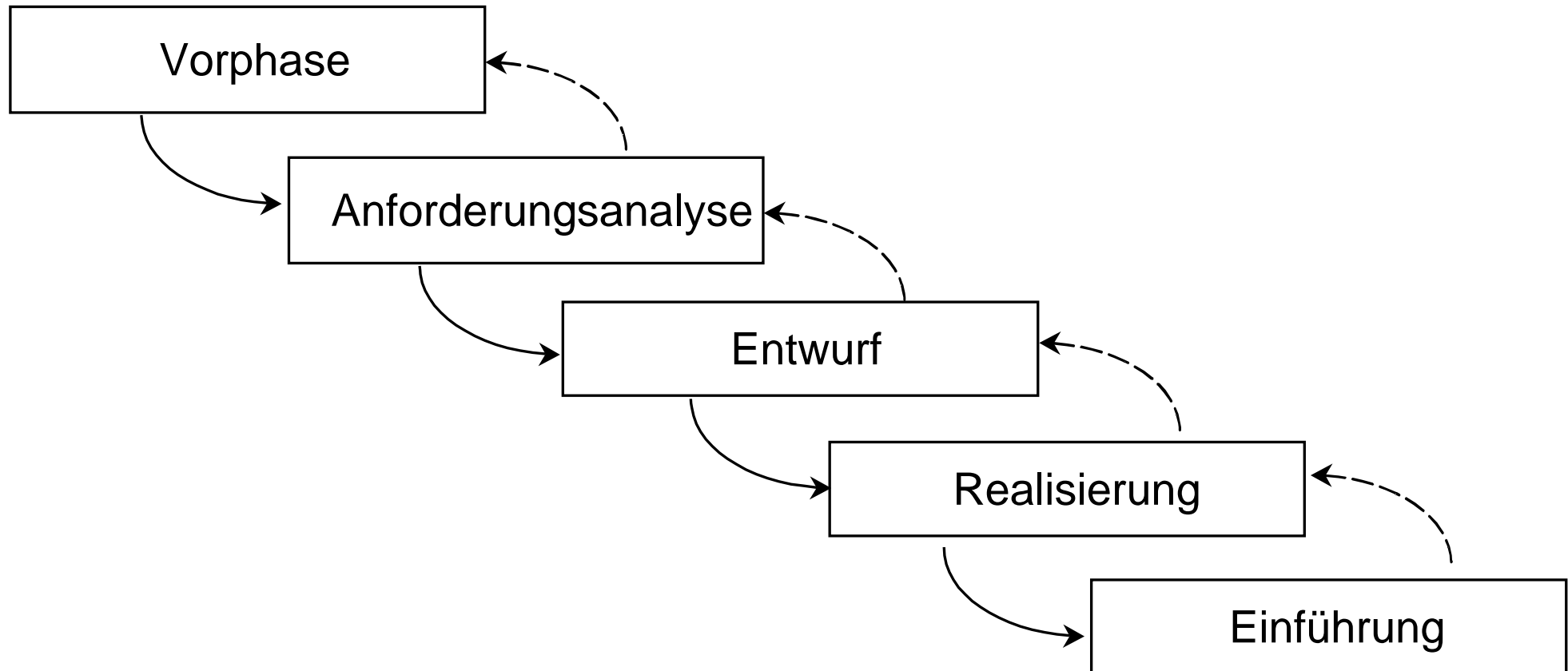
- Welche Teilaufgaben sind in welcher Reihenfolge zur Entwicklung eines Anwendungssystems (AS) zu bewältigen?
- Verwandte Begriffe: Lebenszyklusmodelle, Phasenkonzepte



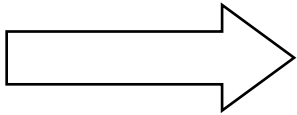
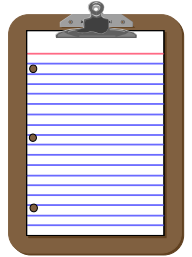
Untergliederung der Entwicklung in "Phasen"

Unschärfe des Begriffs "Phase"

- zeitliche Abfolge (sequentiell, linear sequentiell, parallel, ...)
- oder Aufgabengebiete?

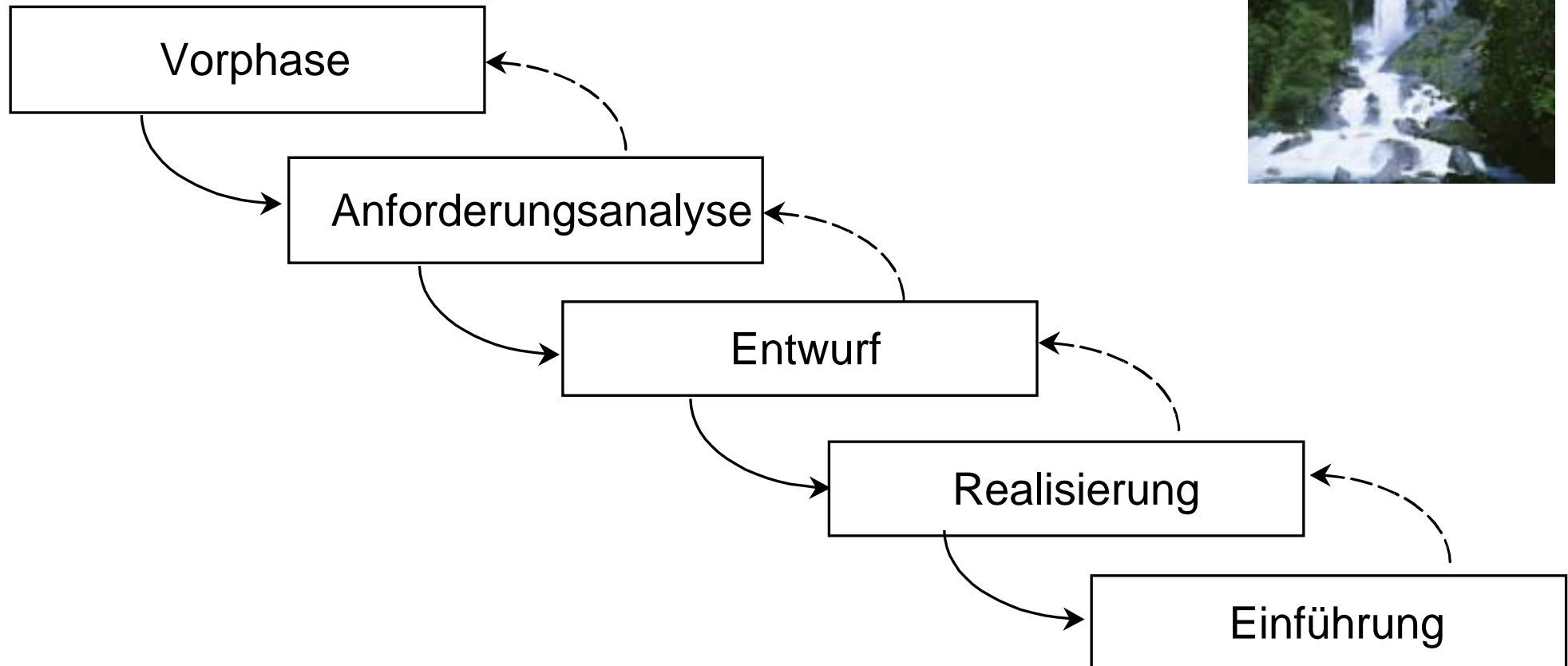


Gliederung



- Einführung
- **Wasserfallmodell**
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Weitere Modelle
- Extreme Programming
- Empirische Befunde

Wasserfallmodell (1/4)

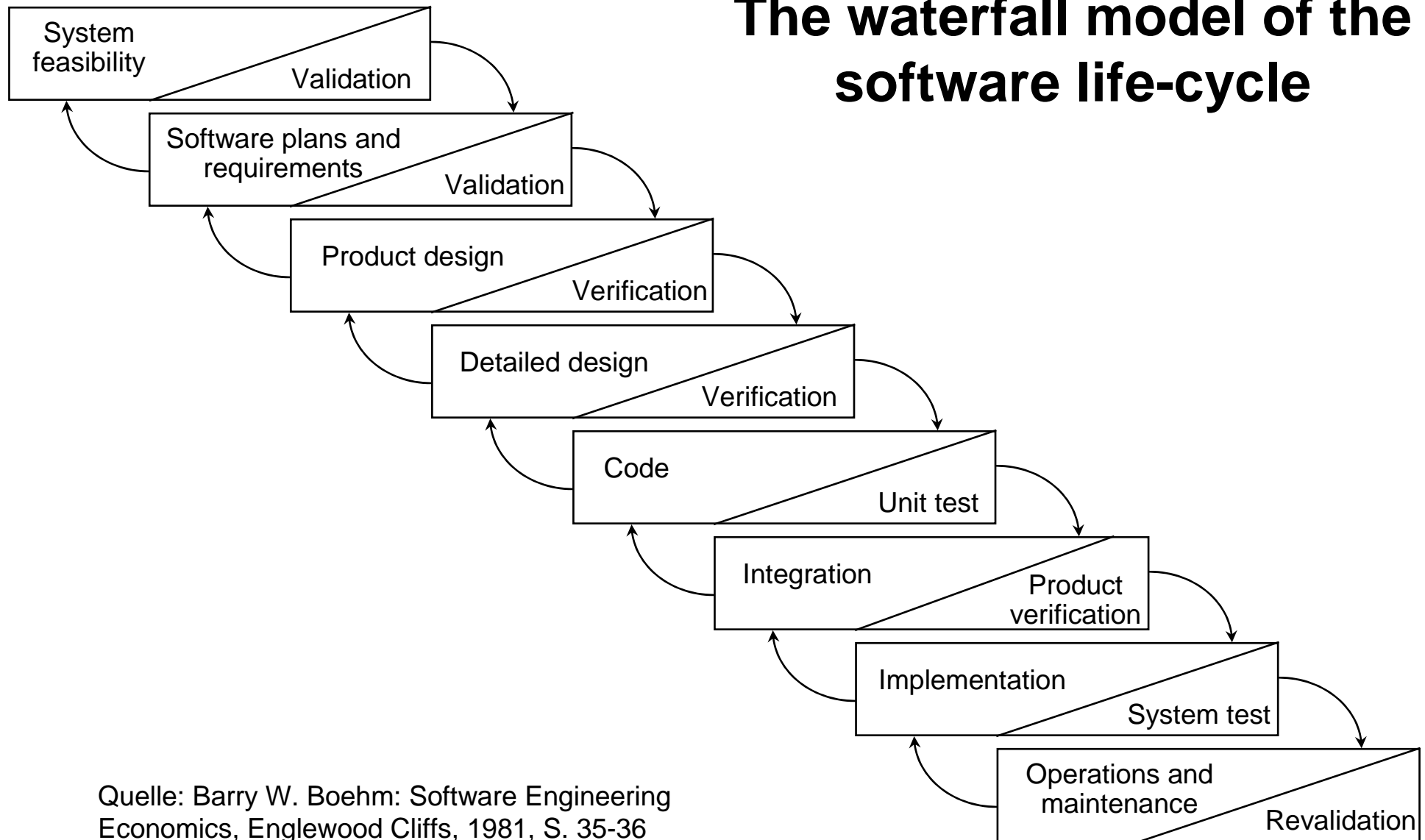


Wasserfallmodell (2/4)



- Zu Beginn jeder Teilaufgabe sind die Anforderungen an diese Teilaufgabe vollständig bestimmt.
- Zum Abschluss einer Teilaufgabe werden Verifizierung und Validierung durchgeführt.
 - Verifizierung (veritas = Wahrheit)
Überprüfung der Übereinstimmung zwischen den spezifizierten und den tatsächlichen Qualitätsmerkmalen (Prüfung gegen die expliziten Vorgaben)
 - Validierung (valere = wert sein, sich eignen für)
Überprüfung der Tauglichkeit eines Gegenstands für seine vorgesehene Aufgabe (Prüfung gegen die Bedürfnisse des Kunden)

The waterfall model of the software life-cycle



Quelle: Barry W. Boehm: Software Engineering Economics, Englewood Cliffs, 1981, S. 35-36

Annahmen und Grenzen des Wasserfallmodells

- Zu Beginn jeder Teilaufgabe sind die Anforderungen an diese Teilaufgabe vollständig bestimmt.
- Frühe Phasen werden mit vollständig entwickelten Dokumenten abgeschlossen.
- Rücksprünge erfordern einen hohen Änderungsaufwand in allen Dokumenten.
- Für bestimmte Softwarekategorien möglicherweise angemessen, z.B.
 - Compiler
 - Sichere Betriebssysteme
- Für die meisten Anwendungssysteme eher unangemessen.



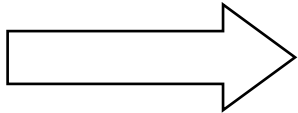
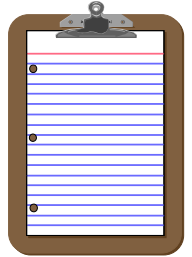
Nennen Sie charakteristische Merkmale des Wasserfallmodells

Für und Wider des Wasserfallmodells

Pro

Contra

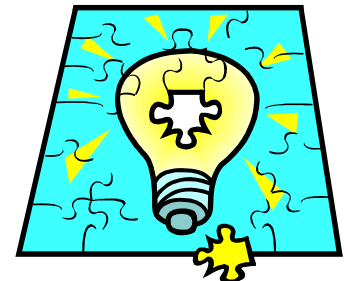
Gliederung



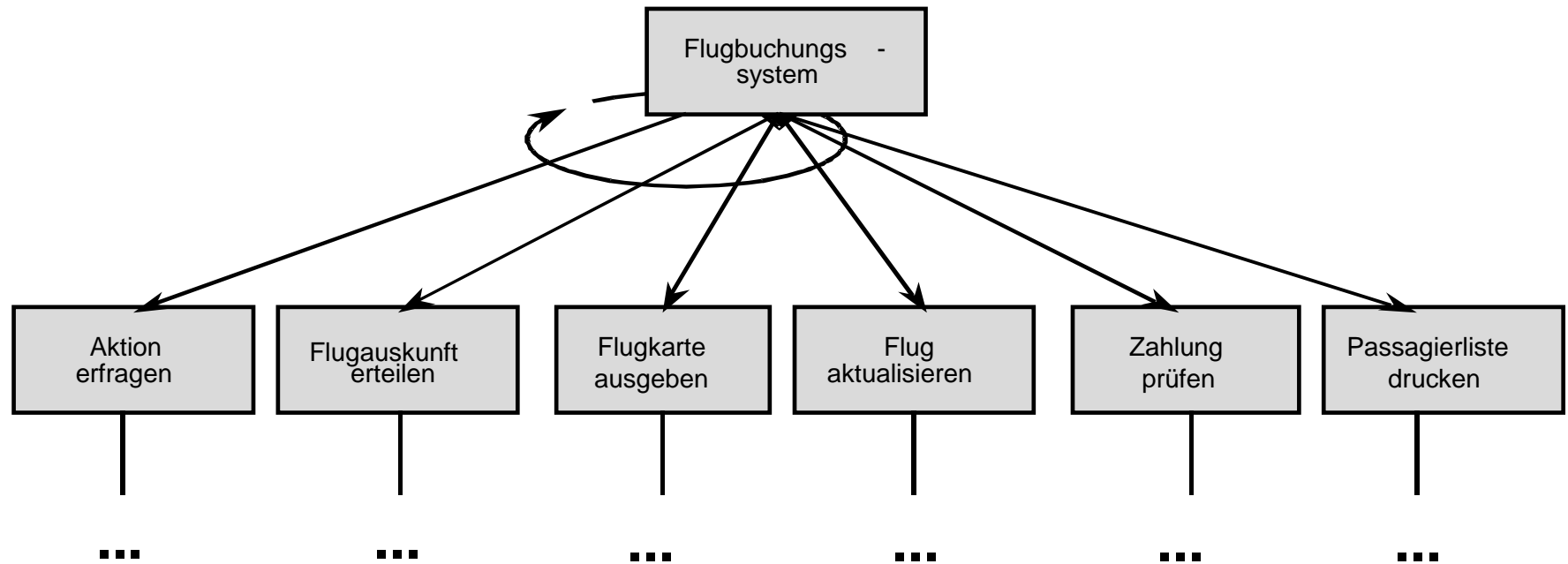
- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Weitere Modelle
- Extreme Programming
- Empirische Befunde

Inkrementelle oder evolutionäre Entwicklung (Prototyping) (1/3)

- **Arbeitsteilung (im Ggs. zum Wasserfallmodell) nicht verrichtungs-, sondern produktorientiert**
- **Entwicklung eines Systemkerns**
 - = entscheidende(s) Element(e) des zu entwickelnden Systems
z. B. Abbildung eines Flugplans und Platzbuchungen in einem Platzreservierungssystem für einen Flugzeugtyp
- **Sukzessive, inkrementelle Ergänzung um weitere Systemelemente**
 - z. B. ... für beliebige Flugzeugtypen, Anschlussreservierungen für Mietwagen, Hotels, etc.

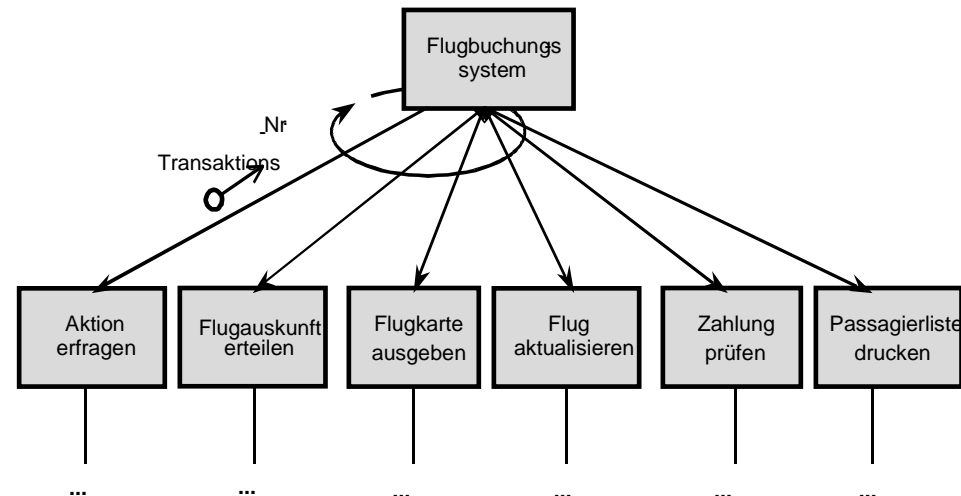


Beispiel: Entwicklung eines Flugbuchungssystems



Eigenschaften der inkrementellen Entwicklung

- „Ich kann Ihnen nicht sagen, was ich will, aber ich sage es Ihnen, wenn ich es sehe.“
- Es werden frühzeitig funktionsfähige Systemteile entwickelt.
- Auftraggeber bekommt frühzeitig einen realistischen Einblick in das Endprodukt
- Problem: Gesamtarchitektur



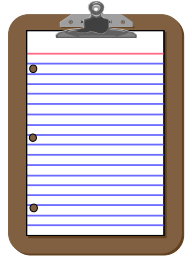
Nennen Sie charakteristische Merkmale der inkrementellen Entwicklung

Für und Wider der inkrementellen Entwicklung

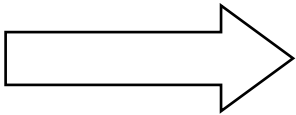
Pro

Contra

Gliederung



- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- **Spiralmodell**
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Weitere Modelle
- Empirische Befunde



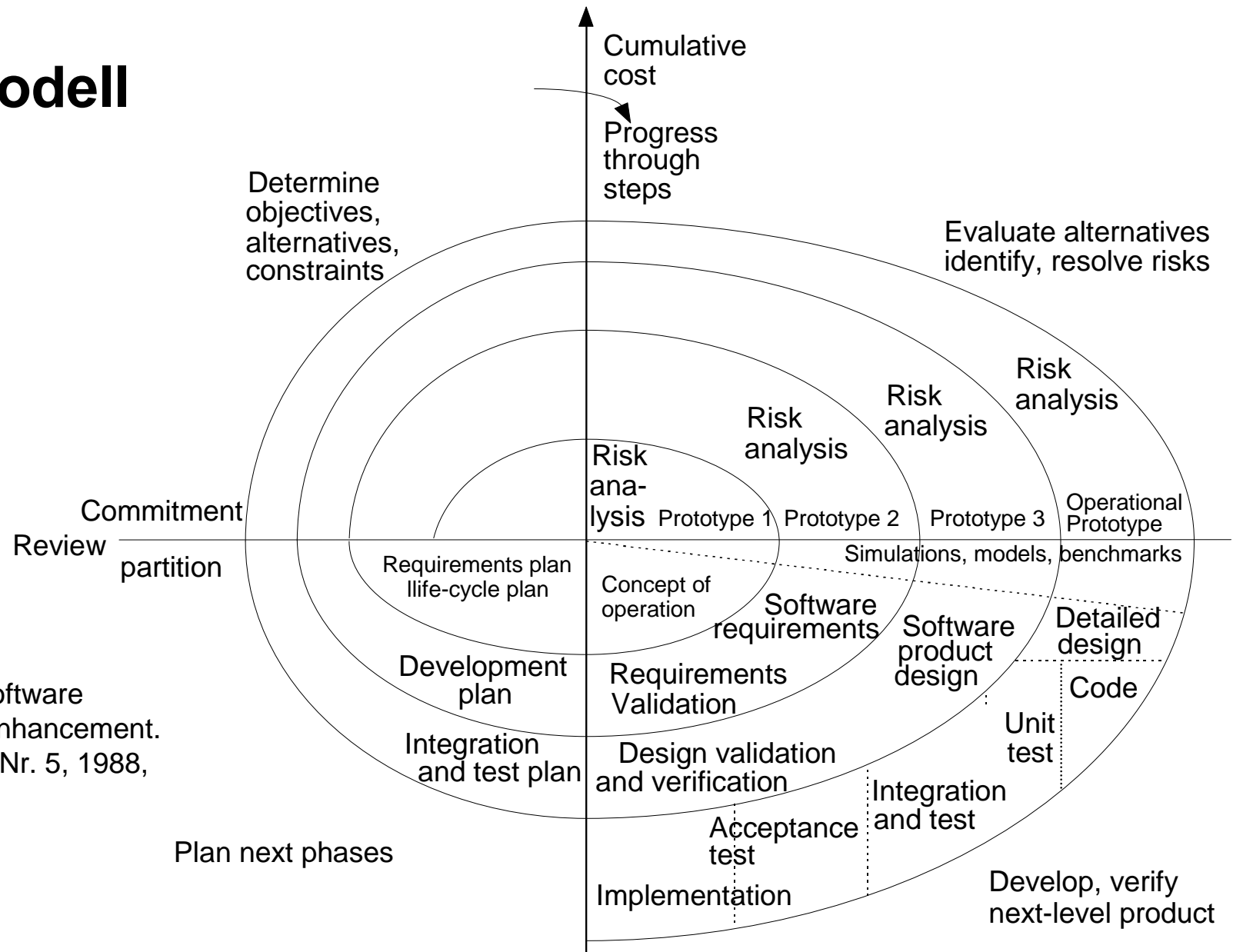
Spiralmodell



- Ablauf orientiert sich an den wirtschaftlichen Risiken des Projektes.
 - Zunächst werden die Produktelemente entwickelt, welche die größten Risiken bergen ...
 - anschließend die weniger risikoreichen
- Mehrfaches Durchlaufen von folgenden Aufgabenbereichen
 - Zielsetzung
 - Risikoanalyse
 - Entwicklung
 - Prüfung und Planung

Quelle: Barry W. Boehm: A Spiral Model of Software Development and Enhancement. In: IEEE Computer. Nr. 5, 1988, S. 61-72

Spiralmodell



Barry W. Boehm:
A Spiral Model of Software
Development and Enhancement.
In: IEEE Computer. Nr. 5, 1988,
S. 61-72

Aufwandsschätzung bei Spiralmodell / inkrementeller Entwicklung

Spiralmodell

- Häufig aktualisierte Aufwandsschätzung im Rahmen der Risikoanalyse jedes neuen Zyklus

Inkrementelle Entwicklung

- Zunächst Schätzung des Gesamtaufwands
- Dann Schätzung des Aufwands für die einzelnen Inkremente;
in der Regel durch die Entwickler selbst

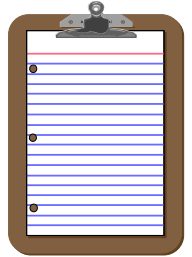
Nennen Sie charakteristische Merkmale des Spiralmodells

Für und Wider des Spiralmodells

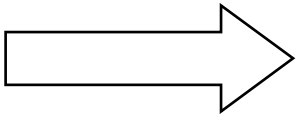
Pro

Contra

Gliederung



- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- **Parallele Entwicklung, Simultaneous Engineering, Concurrent Development**
- Weitere Modelle
- Empirische Befunde



Problem: Anforderungen verändern sich während der Entwicklung

Mögliche Gründe

- Kunde versteht das zu lösende Problem (besser)
- Technische Möglichkeiten verändern sich
- Rahmenbedingungen beim Kunden verändern sich
- ...

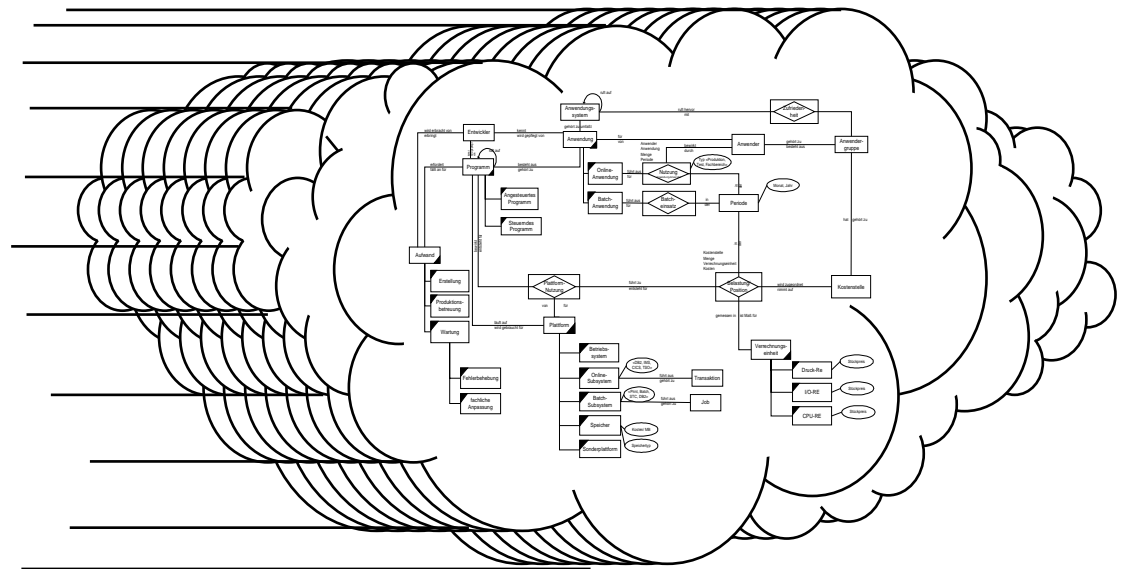
Ressourcen (Personal, Zeit, ...)-Restriktionen verändern sich

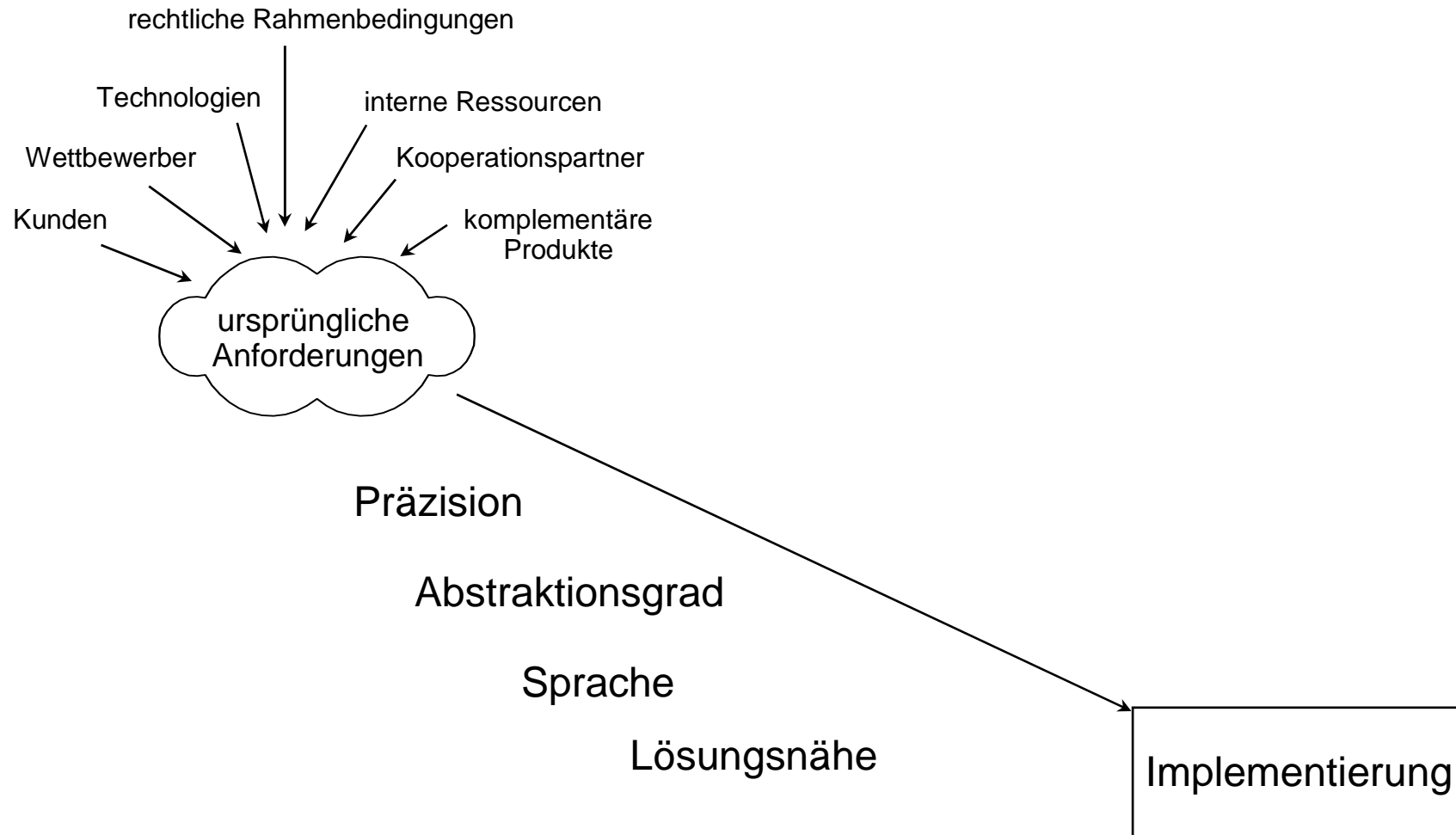
Rahmenbedingungen der Softwareentwicklung

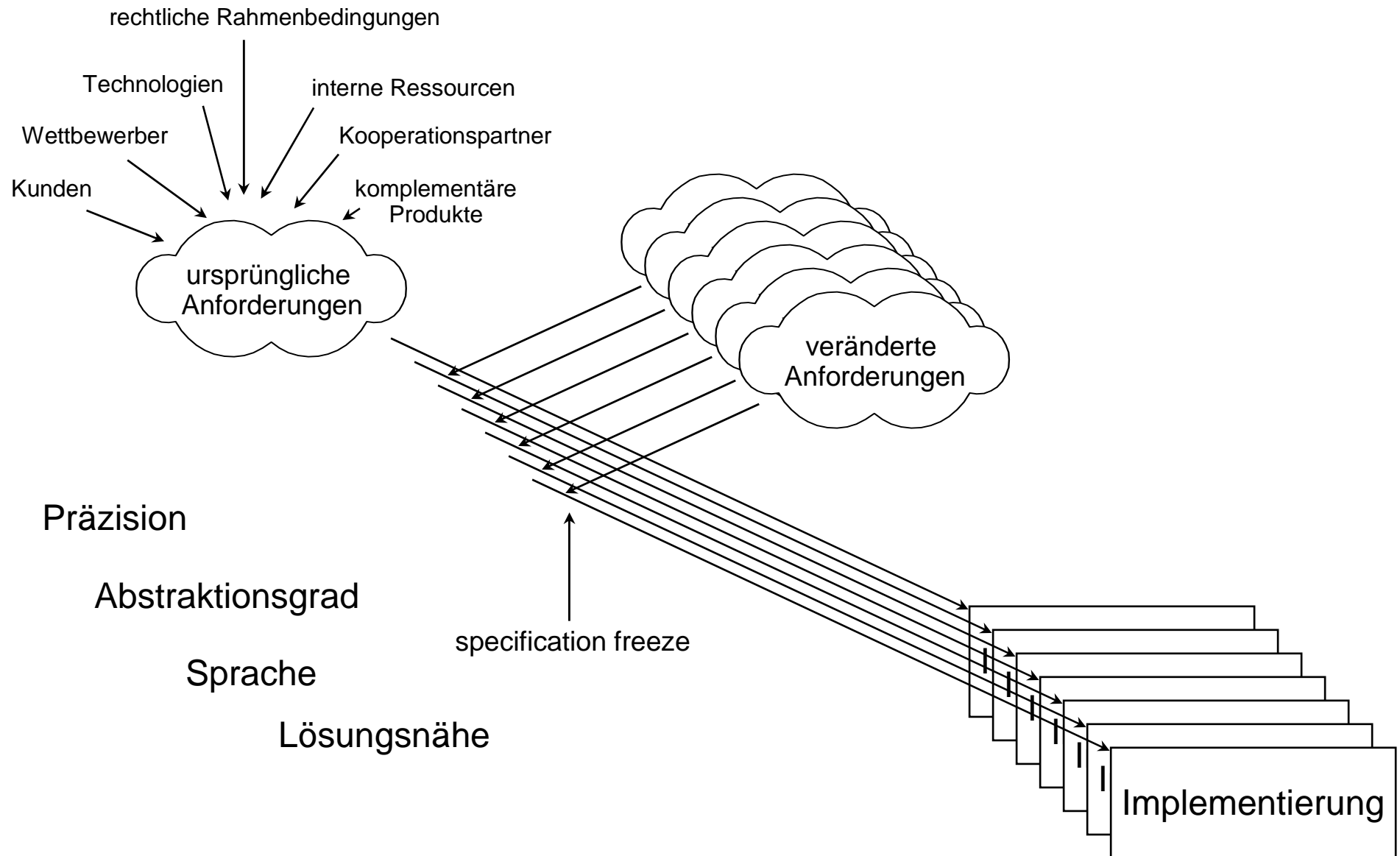
Empirische Befunde:

Es gibt zumindest einige Entwicklungsvorhaben,
in denen die Anforderungen an den Entwicklungsgegenstand

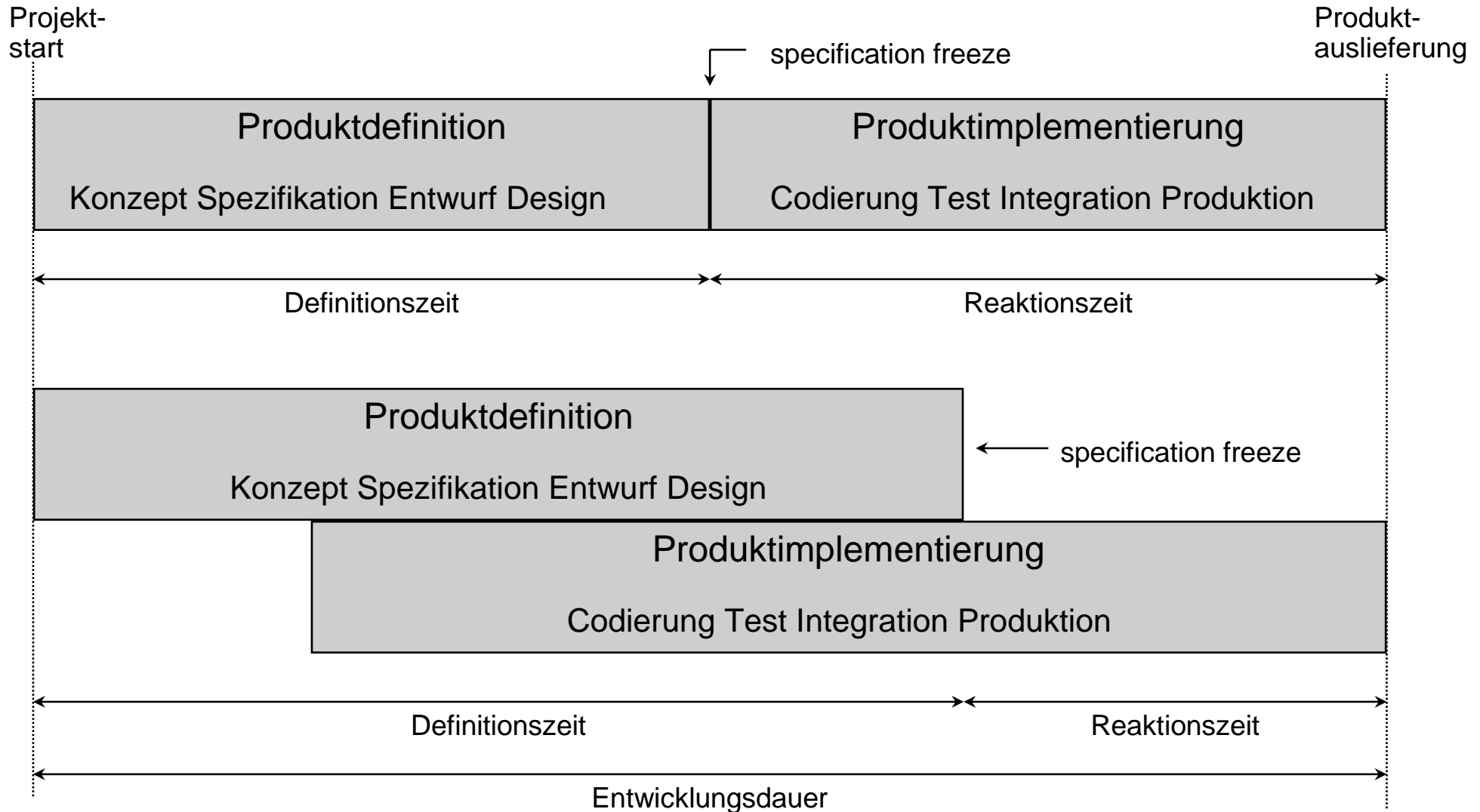
- komplex,
- unbestimmt und
- dynamisch sind.







Parallele Entwicklung, Simultaneous Engineering, ...



Wichtige Ziele des Simultaneous Engineering

Kundennähe

- Ausmaß, in dem Kunden(anforderungen) in das Projekt eingebunden werden

Innovations- /Reaktionsfähigkeit

- Fähigkeit auf geänderte Rahmenbedingungen reagieren zu können

Reaktionsschnelligkeit

- Fähigkeit, möglichst schnell auf (unvorhergesehene) Ereignisse reagieren zu können

Roll-Out-Fähigkeit

- Fähigkeit, das zu entwickelnde Produkt zu (fast) jedem beliebigen Zeitpunkt während der Entwicklung ausliefern zu können

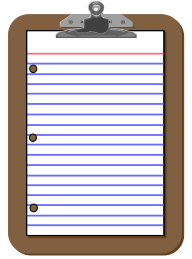
Nennen Sie charakteristische Merkmale der Parallelen Entwicklung etc.

Für und Wider der Parallelen Entwicklung etc.

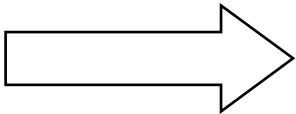
Pro

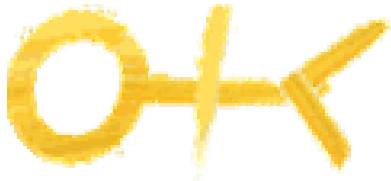
Contra

Gliederung



- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- **Extreme Programming**
- Weitere Modelle
- Empirische Befunde





Extreme Programming (XP)

- Bezeichnung von Kent Beck u.a. für ein „leichtes“ Vorgehensmodell
- Entwickelt in erster Linie für Projekte mit
 - unklaren und sich häufig ändernden Anforderungen
 - bei gleichzeitig hohen Anforderungen des Kunden an die Qualität der Software
- Codezentriertes Prozessmodell, bei dem bestimmte Techniken (= practices) in extremem Maße angewendet werden
- Bestimmte Annahmen und Techniken stehen in krassem Widerspruch zum Software Engineering, wie es z. B. im CMM und der ISO 9000 empfohlen wird.



Motivation des Extreme Programming (XP)

„I observed that people didn't enjoy, and didn't actually use the feedback mechanisms that they read about- synchronized documentation, big testing processes administered by a separate group, extensive and fixed requirements. So I decided to look for feedback mechanisms that

1. people enjoyed, so they would be likely to adopt them,
2. had short-term and long-term benefits, so people would tend to stick to them even under pressure,
3. would be executable by programmers with ordinary skills, so my potential audience was as large as possible and,
4. had good synergistic effects, so we can pay the cost of the fewest possible loops.“

Kent Beck, zitiert nach o. V.: Extreme Programming. 15-Apr-99; <http://ootips.org/xp.html>; Abruf: 2002-10-25



Techniken des XP

- Kleine Releases
- Planungsspiel
- Funktionales Testen
- Modultesten
- Systemmetapher
- Einfacher Entwurf
- Refaktorisierung
- Der Code ist die Dokumentation
- Programmieren in Paaren
- Kollektive Verantwortung für den Code
- Kontinuierliche Code-Integration
- 40-Stunden-Woche
- Kundenvertreter im Team
- Programmierrichtlinien
- Großraumbüros



Kleine Releases

- Hochgradig iterativer Entwicklungsprozess
- Release-Zyklus von 1-3 Monaten
 - Pro Release mehrere Iterationen von 1-3 Wochen Dauer
 - » Pro Iteration Arbeitspakete von 1-3 Tagen
- Nach jeder Iteration kann der Kunde Abweichungen von seinen Wünschen feststellen und korrigieren lassen.
- Das erste Release soll ein funktionierendes Kernsystem sein, welches danach in mehreren Releases inkrementell ausgebaut wird.



Planungsspiel

- Planung der Iterationen
- Ziel: für die nächste Iteration die verfügbaren Ressourcen und die Kundenwünsche in Einklang bringen
- Anforderungen werden als „stories“ („lightweight use cases“ = rudimentäre Anwendungsfälle) auf eine Karteikarte geschrieben
- Kunde versieht die „stories“ mit Prioritäten
- Er bestimmt, welche „stories“ in der kommenden Iteration zu implementieren sind und bis wann die Iteration abgeschlossen sein soll.
- Im Anschluss geben die Entwickler Schätzungen ab, welcher Aufwand für die Implementierung notwendig sein wird.
- Übersteigt der geschätzte Aufwand die verfügbaren Ressourcen, muss der Kunde einige „stories“ auf spätere Iterationen verschieben.
- Dieser Prozess wird so lange wiederholt, bis verfügbaren Ressourcen ausreichen, die Kundenwünsche mit der höchsten Priorität in der nächsten Iteration zu implementieren.



Funktionales Testen (Black-box-Testen)

- Problem: Außer den „stories“ liegt keine Spezifikation des Systems und seiner Bestandteile vor.
- Der Kunde entwickelt Testfälle für seine „stories“, welche dann von einem Tester getestet werden. (Ein Testfall beschreibt eine Menge von Eingabedaten, mit denen ein bestimmter Aspekt des Verhaltens eines Testobjekts geprüft werden soll.)
- Die Entwicklung wird erst fortgesetzt, wenn die Testergebnisse für den Kunden akzeptabel sind.



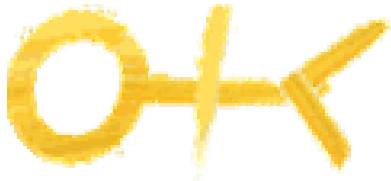
Modultest (Unit Test)

- Die Entwickler entwerfen Testfälle für die von ihnen zu entwickelten Module (bzw. Klassen in der objektorientierten Entwicklung).
- Die Testfälle werden **vor** der Implementierung entwickelt.
- Das zwingt die Entwickler, eine rudimentäre Spezifikation der zu testenden Funktionen in Form von Fallbeispielen zu entwerfen.
- Die Testfälle werden so implementiert, dass sich die Tests jederzeit automatisiert durchführen lassen.
- Die Entwicklung wird erst fortgesetzt, wenn die Modultests vollständig durchgeführt worden sind und keine Fehler mehr aufzeigen.



Systemmetapher

- = eine einfache, und (für Entwickler und Kunden) gut verständliche Geschichte oder ein Bild, welche(s) erklärt, wie das System funktioniert.
- besteht in der Regel aus einigen Elementen, welche den Verarbeitungsablauf metaphorisch darstellen
(z.B. Vergleich mit einer Reise oder einer betrieblichen Aufgabe)
- ersetzt die formale Systemarchitektur
- dient als Leitlinie für die Entwickler



Einfacher Entwurf (1/2)

- Es soll immer nur das entworfen werden, was die im nächsten Schritt zu implementierenden Anforderungen abdeckt.
- Der Entwurf soll so einfach wie möglich werden.
- „the simplest thing that could possibly work“
- Zukünftige Erweiterungen sollen (zunächst) nicht berücksichtigt werden.
- Grund: wenn sich Anforderungen ändern, wird die angelegte Flexibilität möglicherweise nutzlos.
- Sollten später Änderungen nötig werden, wird der Entwurf „refaktorisert“.



Einfacher Entwurf (2/2)

„The right design for the system at any moment is the design that

- runs all the tests,
- says everything worth saying once,
- says everything only once,
- within these constraints contains the fewest possible classes and methods.“

Kent Beck, zitiert nach o. V.: Extreme Programming. 15-Apr-99; <http://ootips.org/xp.html>; Abruf: 2002-10-25



Refaktorisierung

- Aufteilung eines bereits bestehenden Entwurfs in verschiedene Teile unter Beibehaltung der Semantik
- Ziel: Verständlichkeit, Änderbarkeit und Wiederverwendbarkeit des Codes verbessern



Der Code ist die Dokumentation

- XP beschränkt die Dokumentation des implementierten Systems auf den Code
- Code muss selbsterklärend sein
- Dies soll durch eine einfache Struktur und eine treffende Namensgebung erreicht werden.
- Wenn der Code an einer Stelle der Dokumentation bedarf, soll er so lange refaktorisiert werden, bis er auch ohne Kommentar verständlich ist.



Programmieren in Paaren

- Entwurf, Codierung und Test werden von zwei Entwicklern gemeinsam durchgeführt.
- Je ein Entwickler übernimmt die Rolle des Entwicklers, der andere des Prüfers
 - Bildet der Entwurf die „stories“ richtig ab?
 - Kann der Entwurf vereinfacht werden?
 - Bildet der Code den Entwurf richtig ab?
 - Enthält der Code syntaktische oder semantische Fehler?
 - ...
- Paarbildung ist nicht fest, sondern man sucht sich für jedes Arbeitspaket einen geeigneten Partner aus (bis zu vier Mal neu am Tag)
- Nebeneffekt der dynamischen Paarbildung: Wissen über alle Aspekte des Systems verteilt sich über viele Entwickler



Kollektive Verantwortung für den Code

- Für den entstehende Code ist nicht ein bestimmtes Entwicklerpaar alleine verantwortlich, sondern alle Entwickler(paare) im Projekt.
- Jedes Entwicklerpaar darf jederzeit überall Änderungen vornehmen, um z.B. die Verständlichkeit des Codes zu erhöhen.
- Jedes Paar muss nach jeder Änderung alle Tests durchführen.



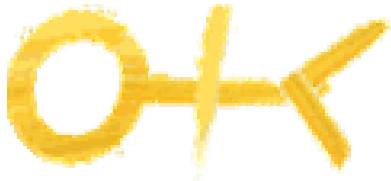
Kontinuierliche Code-Integration

- Neu entwickelter oder geänderter Code wird in regelmäßigen Abständen (maximal nach einem Tag) in die aktuelle „Code-Basis“ (baseline) integriert.
- Zur Integration dient ein spezieller Integrationsrechner.
- Dort spielt ein Paar seine Entwicklungen / Änderungen ein und führt alle Tests aus.
- Treten dabei Fehler auf, muss das Paar seine Entwicklungen / Änderungen zurücknehmen und dafür sorgen, dass alle Fehler behoben werden.
- Dadurch ist immer ein lauffähiges System vorhanden.



40-Stunden Woche

- Das Programmieren in Paaren stellt hohe Anforderungen an die Entwickler.
- Beide müssen jederzeit konzentriert bei der Sache sein.
- Daher werden bei XP geregelte Arbeitszeiten gefordert (40 Stunden / Woche) als Richtwert.
- Überstunden (= unfreiwillige Mehrarbeit) sollen nur in Ausnahmefällen gemacht werden.



Kundenvertreter im Team

- Da keine umfassende Spezifikation vorliegt, gibt es viele Rückfragen an den Kunden.
- Ein Vertreter des Kunden muss permanent für die Entwickler verfügbar sein.
- Dies soll ein zukünftiger Anwender des Systems sein.
- Aufgaben des Kundenvertreters:
 - Er steuert die Entwicklungsrichtung.
 - Er klärt Unklarheiten.
 - Er setzt Prioritäten.
 - Er entwickelt die Testfälle für die funktionalen Tests



Programmierrichtlinien

- Um das Arbeiten in Paaren und die kollektive Verantwortung für den Code zu erleichtern, halten sich die Entwickler an feste Programmierrichtlinien, z.B.
 - Art der Namensgebung für Klassen und Variablen
 - Formatierung des Codes
 - ...
- Diese werden von den Entwicklern selbst vereinbart und strikt eingehalten.
- Dadurch entsteht einheitlich strukturierter und formulierter Code, der von allen verstanden und geändert werden kann.



Großraumbüros

Offene Arbeitsatmosphäre

- Beugt dem „Einzelkämpfertum“ vor.
- Erleichtert spontane Kommunikation.
- Erleichtert das „Hilferufen“.



Voraussetzungen des XP



Für und Wider des XP

Pro

Contra

Quellen zum XP im WWW

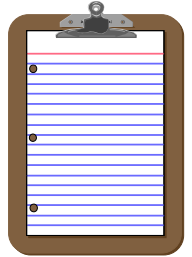
- <http://ootips.org/xp.html>
- <http://www.xprogramming.com/>
- <http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap>

Literaturhinweise zum XP

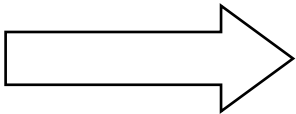


- Kent Beck: Extreme Programming Explained: Embrace Change. Reading, Addison-Wesley. 1999
- Kent Beck: Embracing Change with Extreme Programming. In: Computer. Nr. 10, 1999, S. 70-79.
- Ralf Reißing: Extremes Programmieren. In: Informatik Spektrum. Nr. 2, 2000, S. 118-121.
- Hans Wegener: Extreme Ansichten - Für und Wider des Extreme Programming. In: IX : Multiuser - Multitasking Magazin. Nr. 12, 1999, S. 126-130.

Gliederung



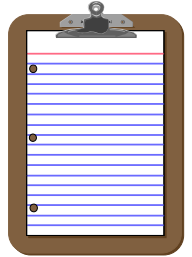
- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Extreme Programming
- Weitere Modelle
- Empirische Befunde



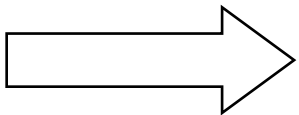
Weitere Modelle

- Rapid Application Development (RAD)
http://sysdev.ucdavis.edu/WEBADM/document/rad_toc.htm
- Adaptive Software Engineering
- Dynamic Systems Development Method (DSDM)
<http://www.dsdm.org/>
- Crystal
<http://www.crystallmethodologies.org/>
- ...

Gliederung



- Einführung
- Wasserfallmodell
- Inkrementelle oder evolutionäre Entwicklung (Prototyping)
- Spiralmodell
- Parallele Entwicklung, Simultaneous Engineering, Concurrent Development
- Extreme Programming
- Weitere Modelle
- Empirische Befunde



Weltz, Ortmann: Das Softwareprojekt (1992)

Untersuchungsmethode und –gegenstand

- Standardisierte schriftliche Befragungen in 46 Softwareentwicklungsprojekten
 - davon ca. 75 % Neuentwicklungen und
 - ca. 85 % mit einer Projektdauer von mehr als zwei Jahren
- In welchen Unternehmen fanden die Projekte statt?
 - 48 % in Anwenderunternehmen,
 - 46 % in Softwarehäusern und
 - 7 % bei DV-Herstellern

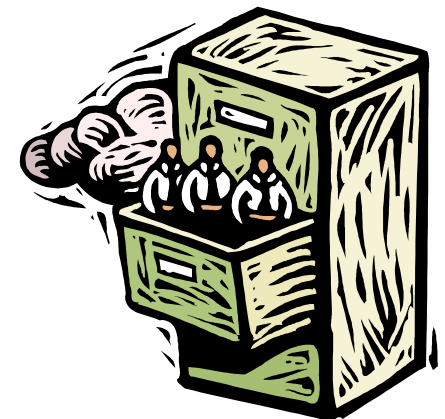


Friedrich Weltz, Rolf G. Ortmann: Das Softwareprojekt: Projektmanagement in der Praxis. Frankfurt u. a. 1992

Ergebnisse zur Projektorganisation

Weltz, Ortmann: Das Softwareprojekt (1992)

- Nur ca. 50 % der Projekte verfügen bei Projektbeginn über eine explizite Projektorganisation.
- Nur ca. 25 % der Projekte haben zu Beginn eine feste Projektorganisation und halten diese im Projektverlauf bei.
- In ca. 20 % der Projekte (vorwiegend kleine Projekte) gibt es überhaupt keine formale Projektorganisation.
- In 40 % der Projekte wird die Projektorganisation im Verlauf der Projekte ein- oder mehrmals verändert.



Ergebnisse zur Arbeitsteilung

Weltz, Ortmann: Das Softwareprojekt (1992)

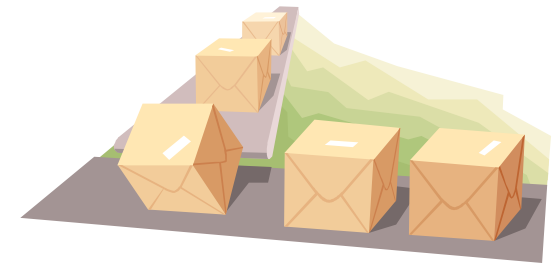
- Nur in ca. 20 % der Projekte erfolgt eine an den Projektphasen ausgerichtete Arbeitsteilung.
- In ca. 60 % der Projekte erfolgt eine Arbeitsteilung nach Subsystemen der zu entwickelnden Software.
- In ca. 40 % der Projekte gibt es zusätzlich Spezialisten, die für bestimmte Systemfunktionen zuständig sind.
- Ähnlich wie die Projektorganisation wird auch die Arbeitsteilung oft erst im Projektverlauf „erfunden“.



Ergebnisse zur Ablauforganisation

Weltz, Ortmann: Das Softwareprojekt (1992)

- Feste verbindliche Muster für die Ablaufgestaltung fehlen in vielen Projekten.
- Ca. 60 % der Projekte gliedern den Entwicklungsprozess durch die sukzessive Bearbeitung von Subsystemen.
- Ca. 20 % der Projekte gliedern den Entwicklungsprozess mit Hilfe eines Phasenschemas:
 - Nur in ca. 24 % dieser Projekte wurde die Phaseneinteilung auch eingehalten.
 - In ca. 40 % dieser Projekte kam es zu Überschneidungen der Phasen.
 - In 13 % der Projekte kam es häufig zu Rücksprüngen in vorherige Phasen.
 - In 22 % der Projekte wurde der Verlauf ohne erkennbare Systematik von aktuellen Bedingungen und von Zufällen bestimmt.



Art der Aufgaben

Weltz, Ortmann: Das Softwareprojekt (1992)

- Die zu bewältigenden Aufgaben gehören nur zum Teil zur Softwareentwicklung im engeren Sinne. Viele erfolgsrelevante Aufgaben gehören nicht zum offiziellen Arbeitsprogramm, z. B.:
 - Änderung, Korrektur und Ergänzung unsinniger, widersprüchlicher oder unvollständiger Anforderungen
 - Neustrukturierung der Aktivitäten, die sich auf diese Anforderungen beziehen
 - Herstellung von Konsens unter den Beteiligten
 - Anpassung von Vorgehensregelungen an die Erfordernisse des Projektes

Kontroll- und Verständnisfragen



- Mit Hilfe welcher Eigenschaften lassen sich Vorgehensmodelle unterscheiden?
- Beschreiben Sie die wesentlichen Unterschiede zwischen der inkrementellen und der parallelen Entwicklung.
- Worin liegt der Unterschied zwischen Verifizierung und Validierung?
- Sie sollen die Ablaufplanung für ein Projekt bestimmen, für das absehbar ist, dass sich die Kundenanforderungen erst während der Laufzeit vollständig entwickeln werden. Welches Vorgehensmodell wählen Sie? Begründen Sie Ihre Wahl!
- Diskutieren Sie die These: „Je schlechter die Zeiten, desto leichter die Vorgehensmodelle.“
- Beschreiben Sie die wesentlichen Praktiken des Extreme Programming mit Fachbegriffen der Organisationslehre.



Literaturhinweise

- Helmut Balzert: Lehrbuch der Software-Technik. Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg – Berlin, 1998, S. 97-138
- Barry W. Boehm: A Spiral Model of Software Development and Enhancement. In: IEEE Computer. Nr. 5, 1988, S. 61-72
- Barry W. Boehm: Software Engineering Economics, Englewood Cliffs, 1981, S. 35 ff.
- Gustav Pomberger: Methodik der Softwareentwicklung. In: Karl Kurbel, Horst Strunz (Hrsg.): Handbuch der Wirtschaftsinformatik. Stuttgart 1990, S. 215-236
- Dietrich Seibt: Systemlebenszyklus, Management des. In: Peter Mertens u.a. (Hrsg.): Lexikon der Wirtschaftsinformatik. 3. Aufl., Berlin - Heidelberg - New York 1997, S. 393-395

Quellen WWW

- Gesellschaft für Informatik. Fachgruppe 5.11: Vorgehensmodelle für die betriebliche Anwendungsentwicklung. <http://www.vorgehensmodelle.de/>
- o.V.: Entwicklungsstandard für IT-Systeme des Bundes: <http://www.vorgehensmodell.de>
- Kent Beck u.a.: Manifesto for Agile Software Development. o.O. 2001 <http://www.agilemanifesto.org/>