SOFTWARE ENGINEERING

2
Modelle zur Organisation von Software-Entwicklungsprozessen

JOHANNES KEPLER UNIVERSITÄT LINZ

INSTITUT FÜR WIRTSCHAFTSINFORMATIK

Prozessmodelle

SOFTWARE ENGINEERING

Prozessmodelle dienen zur Benennung und Ordnung von Tätigkeiten im Rahmen der Softwareentwicklung.

In der Literatur sind dazu viele Vorschläge zu finden, von denen hier nur eine Auswahl vorgestellt wird.

Grundlage für die Auswahl ist der Verbreitungsgrad und die Praxisrelevanz der Prozessmodelle sowie das Bestreben, den Studierenden mehrere unterschiedliche Ansätze darzulegen.

Das klassische sequenzielle Phasenmodell

SOFTWARE ENGINEERING

Phasenmodelle existieren in zahlreichen Variationen.

Sie postulieren im Wesentlichen folgende Phasen des Software-Entwicklungsprozesses:

- Problemanalyse und Grobplanung
- Systemspezifikation und Planung
- System- und Komponentenentwurf
- Implementierung und Komponententest
- System- und Integrationstest
- Betrieb und Wartung

Das klassische sequenzielle Phasenmodell

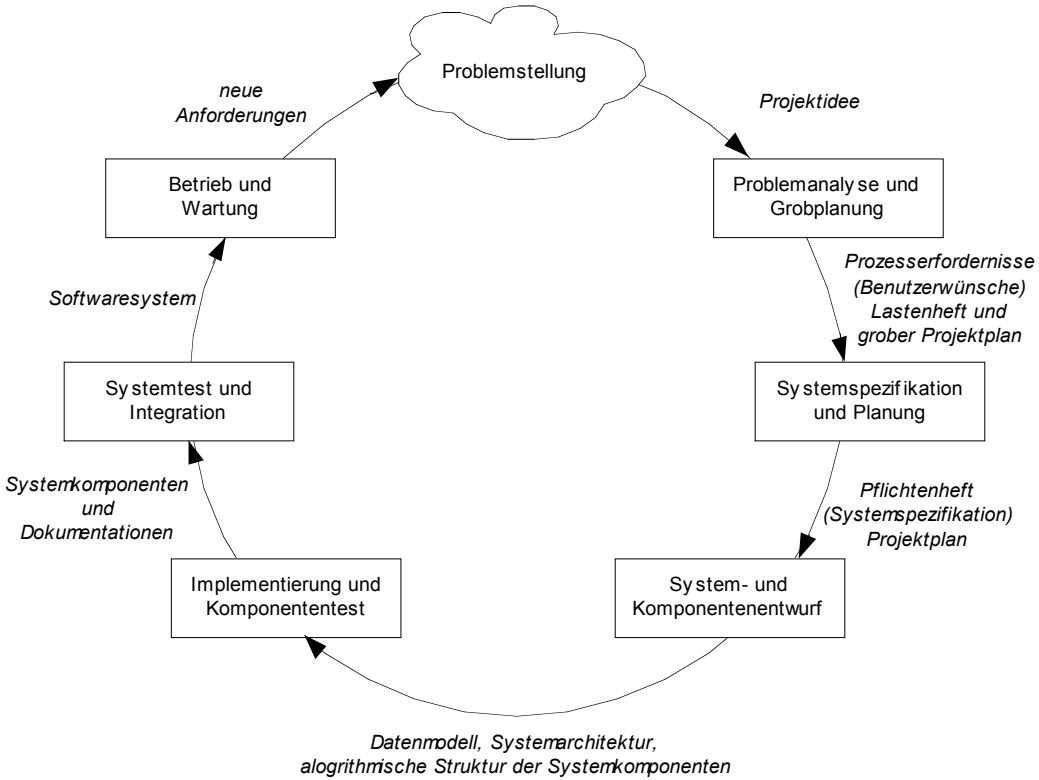
SOFTWARE ENGINEERING

Die dem Phasenmodell zugrunde liegende Vorgehensweise bei der Software-Entwicklung beruht auf dem Prinzip

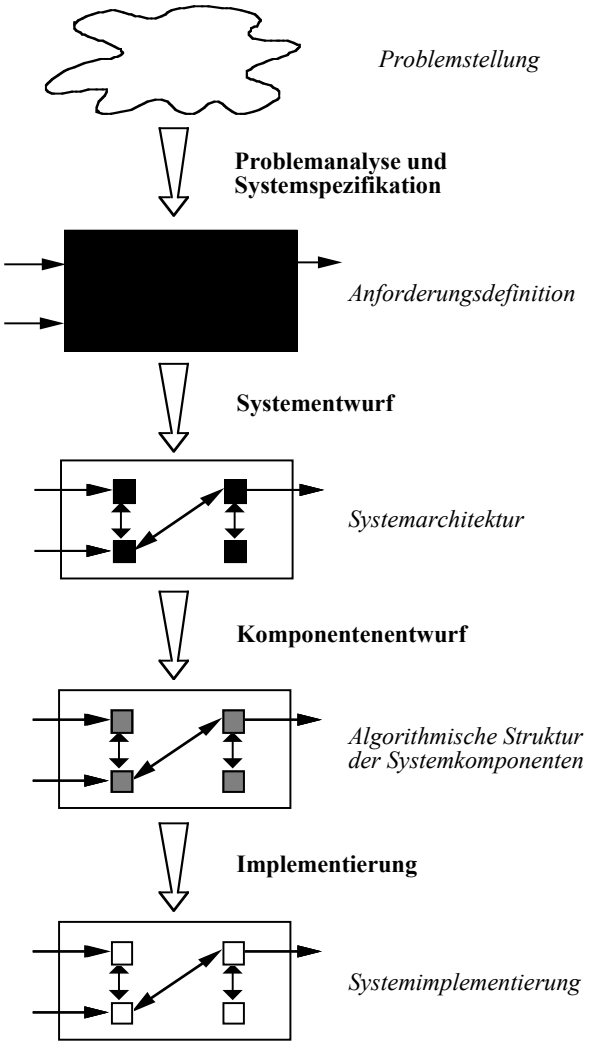
- dass für jede der Phasen klar zu definieren ist, welche Ergebnisse erzielt werden müssen
- und dass eine Phase erst dann in Angriff genommen werden darf, wenn die vorhergehende vollständig abgeschlossen ist.

Die Anwendung dieser streng sequenziellen Vorgangsweise soll dazu führen, dass Softwareprojekte besser planbar, organisierbar und kontrollierbar werden.

Das klassische sequenzielle Phasenmodell



Sukzessive Dekomposition



Vor- u. Nachteile des klassischen sequenziellen Phasenmodells

SOFTWARE ENGINEERING

Vorteile

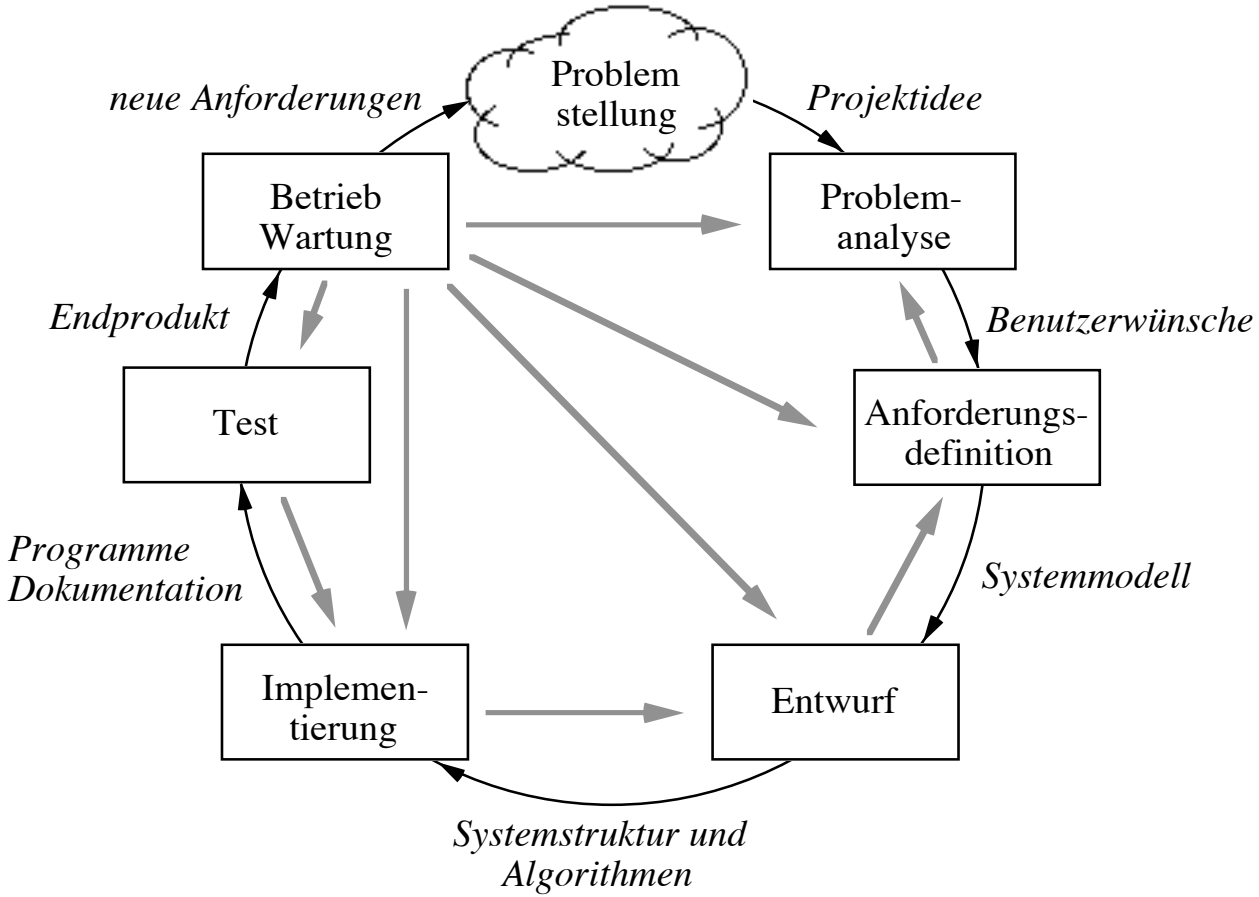
- klarer Rahmen, exakt definierte Tätigkeiten, präzise Abgrenzung zwischen den Phasen
- unabhängig vom Anwendungsgebiet und von der Projektgröße
- strukturierter Entwicklungsprozess fördert Strukturiertheit des Produkts
- ermöglicht arbeitsteiligen Entwicklungsprozess

Nachteile

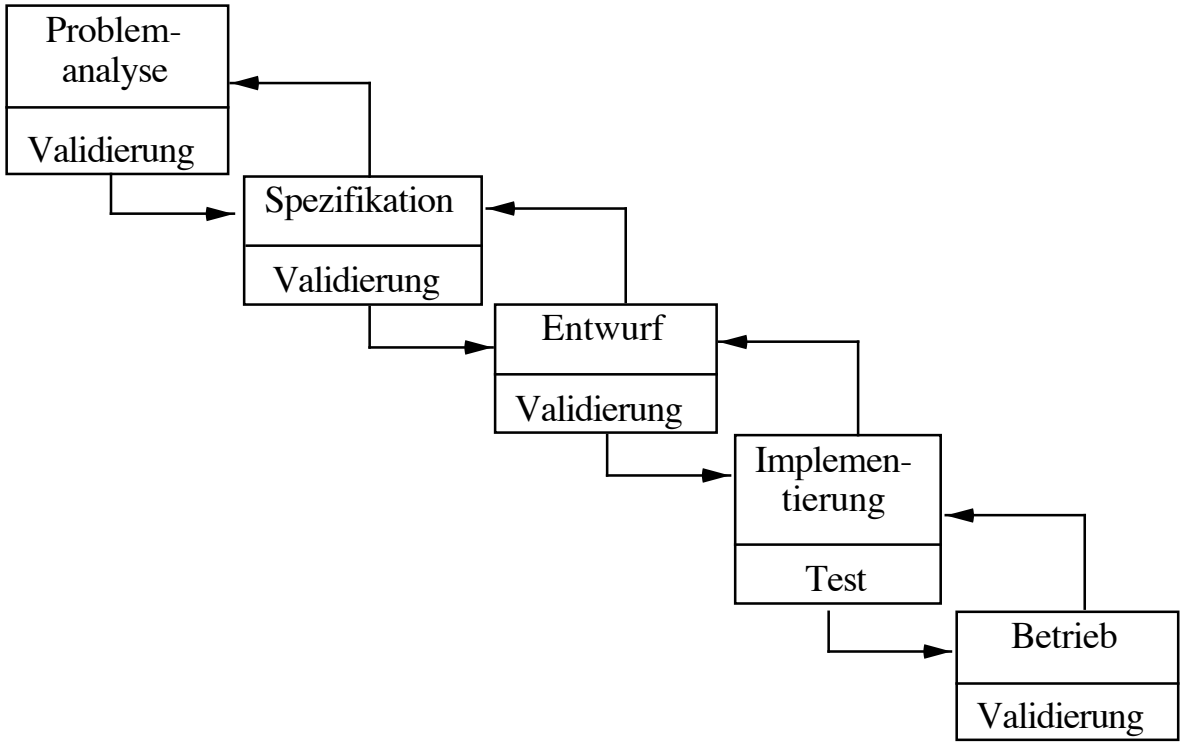
- Linearisierung ist unrealistische Idealisierung
- Vollständigkeit der Phasenergebnisse (Zwischenprodukte) selten erreichbar
- präzise Phasentrennung steht im Widerspruch zur Realität
- Fehler werden spät erkannt ==> hoher Änderungsaufwand

Phasenmodell mit Rückkopplungen

SOFTWARE ENGINEERING



Das Wasserfallmodell



Das V-Modell

SOFTWARE ENGINEERING

Das V-Modell ist eine Erweiterung des Wasserfall-Modells.

Es gliedert den Software-Entwicklungsprozess (ähnlich dem Wasserfallmodell)

- in eine Sequenz von Phasen
- und integriert explizit den Qualitätssicherungsprozess

und zwar so, dass eine klare Zuordnung der Phasen des Qualitätssicherungsprozesses zu den Phasen des Entwicklungsprozesses in Form eines V vorgenommen wird.

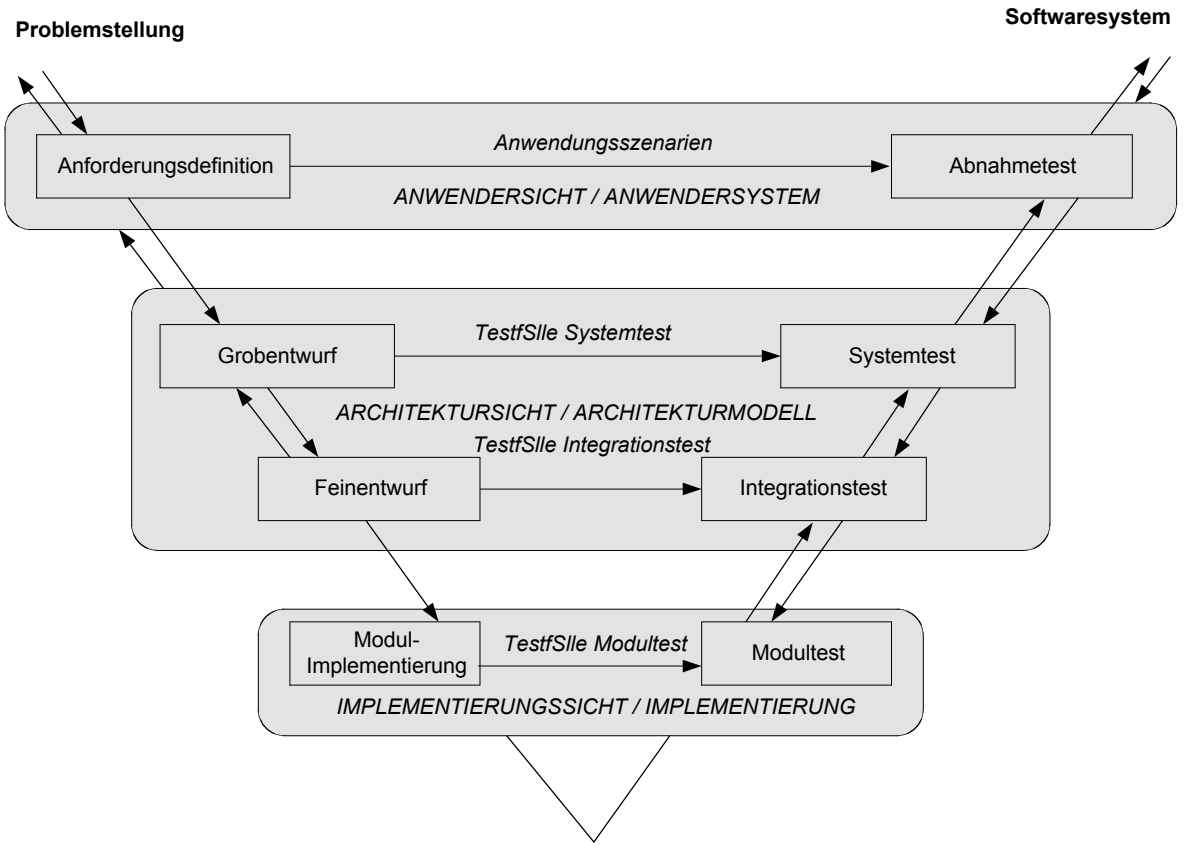
Das V-Modell

SOFTWARE ENGINEERING

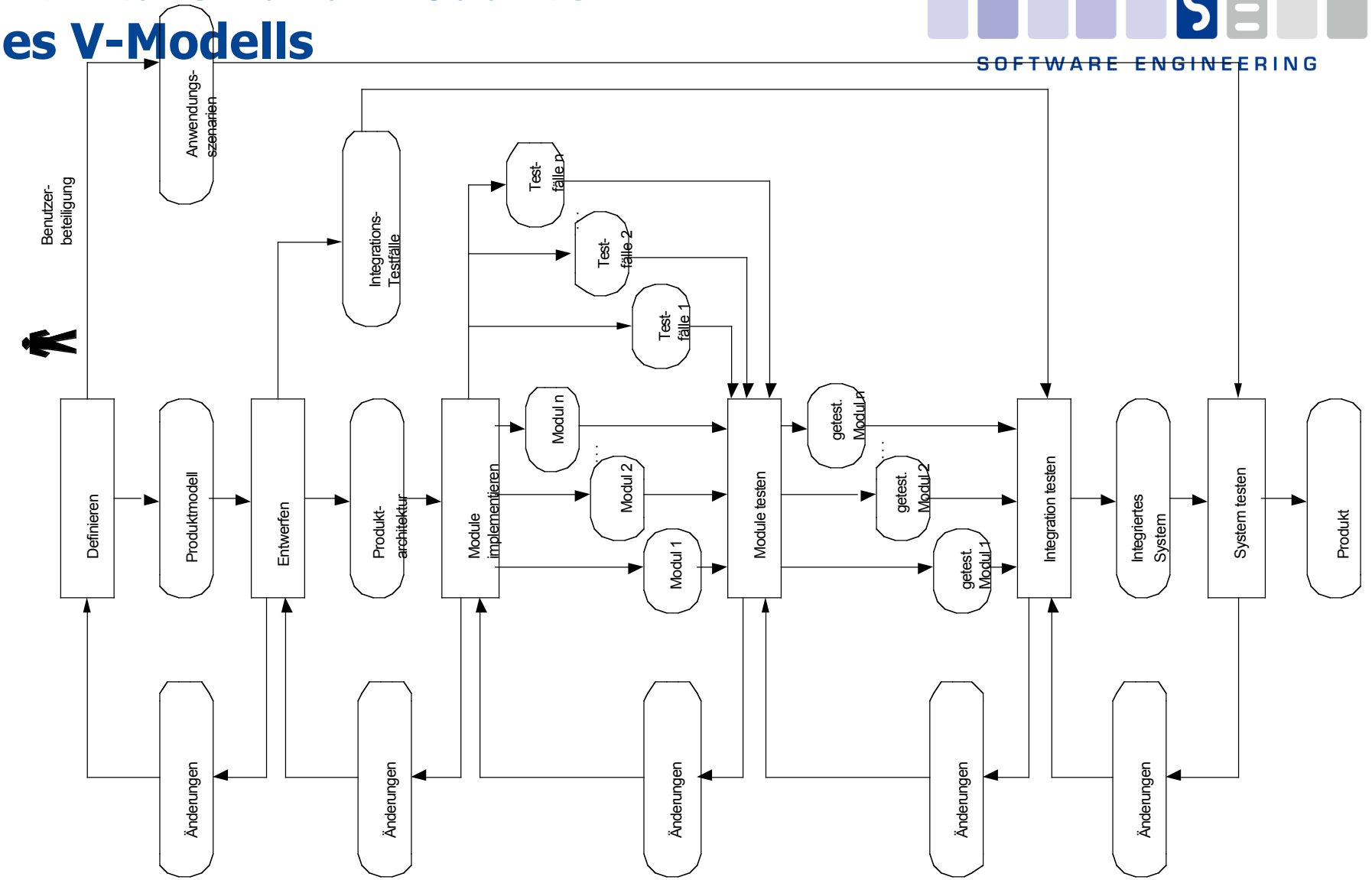
Das Modell strukturiert den Software-Entwicklungsprozess in drei Sichten:

- Die *Anwendersicht*, die sich in Form eines Anwendungssystems bestehend aus einer Systemspezifikation und dem abgenommenen Softwareprodukt manifestiert.
- Die *Architektursicht*, die sich in Form eines Architekturmodells bestehend aus den Spezifikationen von Teilsystemen, den dazu entworfenen Systemarchitekturen, den getesteten Teilsystemen und dem getesteten Gesamtsystem manifestiert.
- Die *Implementierungssicht*, die sich in Form der Implementierung bestehend aus den Modulspezifikationen und den getesteten Modulen manifestiert.

Das V-Modell (3)



Aktivitäten und Produkte des V-Modells



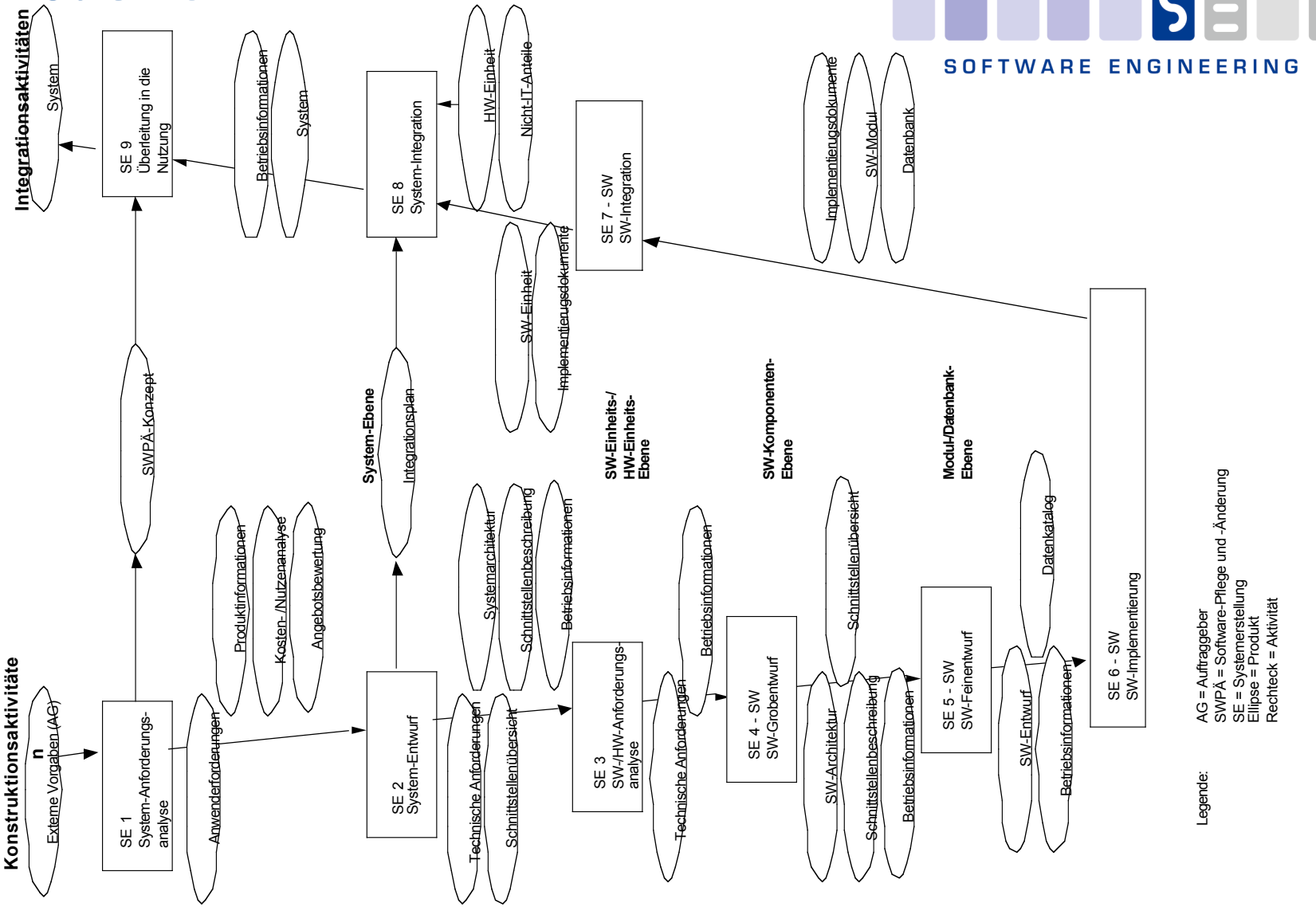
Gliederung des V-Modells

SOFTWARE ENGINEERING

Das V-Modell ist in vier Teilmodelle gegliedert:

- Teilmodell SE: Systemerstellung
- Teilmodell QS: Qualitätssicherung
- Teilmodell KM: Konfigurationsmanagement
- Teilmodell PM: Projektmanagement

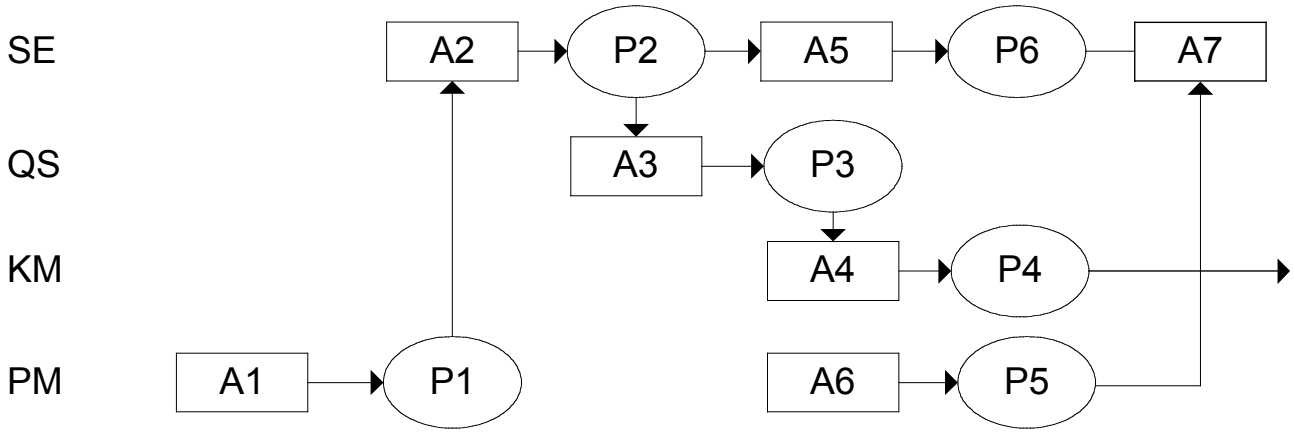
Teilmodell SE



Quelle: Balzert 1998

Zusammenwirken der Teilmodelle

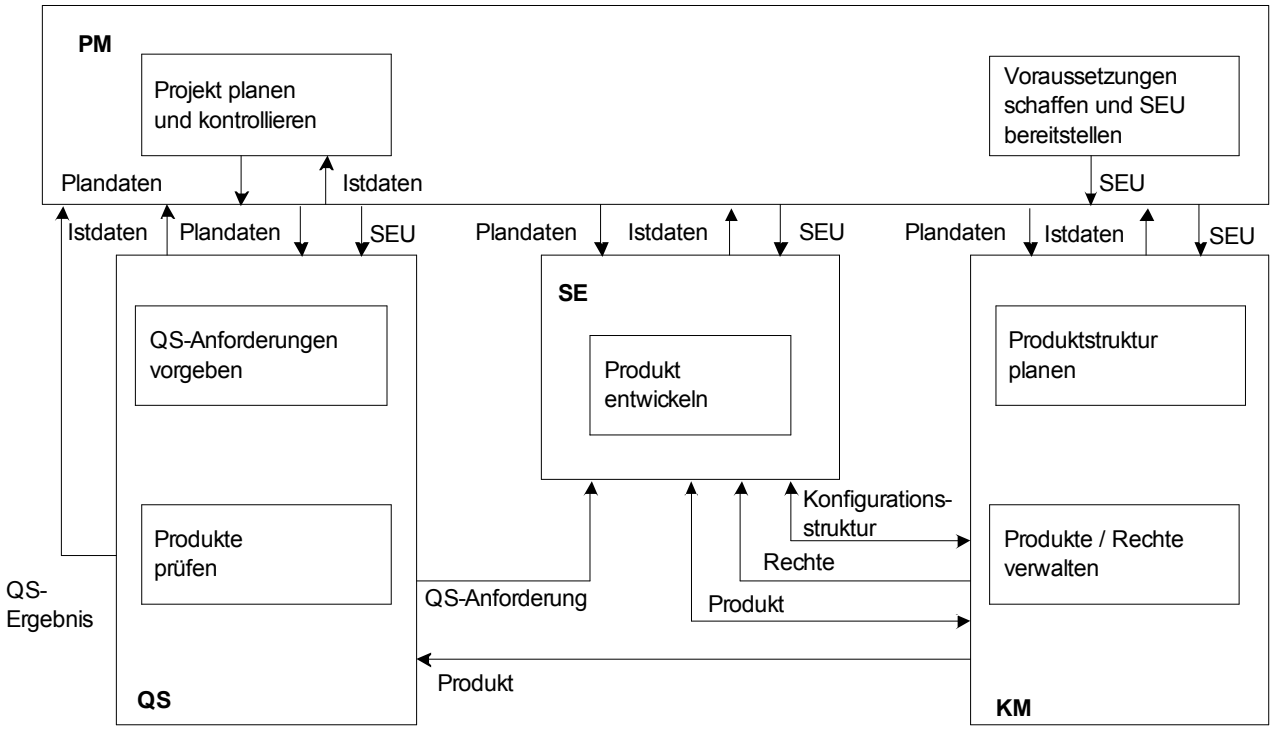
SOFTWARE ENGINEERING



Ai = Aktivitäten, Pi = Produkte

Aktivitäten- und Produktflusses über die 4 Teilmodelle SE, QS, KM, PM (Quelle: Hansen und Neumann 2001)

Zusammenwirken der Teilmodelle



Legende:
SEU = Software-Entwicklungs-Umgebung
PM = Teilmodell Projektmanagement
SE = Teilmodell Systemerstellung
QS = Teilmodell Qualitätssicherung
KM = Teilmodell Konfigurationsmanagement

Quelle: Balzert 1998

Vor- und Nachteile des V-Modells

SOFTWARE ENGINEERING

Vorteile

- » äquivalent zum klassischen sequenziellen Vorgehensmodell
- » Integration und Detaillierung der Teilmodelle Systemerstellung, Qualitätssicherung, Konfigurations- und Projektmanagement

Nachteile

- » äquivalent zum klassischen sequenziellen Vorgehensmodell
- » zu aufgebläht, zu direktiv und zu wenig methodenneutral
- » Verlust an Flexibilität
- » beträchtlicher (bürokratischer) Mehraufwand (dessen Nutzen nicht unumstritten ist)

Konsequenzen aus den Erfahrungen mit klassischen Phasenmodellen

SOFTWARE ENGINEERING

Das *Ziel* der Systementwicklung ist nicht völlig unabhängig vom *Lösungsweg* erreichbar

→ Wechselwirkungen zwischen *Ziel* und *Weg* in die Entwicklungsmethodik einbeziehen

→ Prototypingorientierte Entwicklungsmethodik

Das prototyping-orientierte Prozessmodell

Gegenwärtige Situation

SOFTWARE ENGINEERING

- keine allgemein akzeptierten Begriffe für Prototyp und Prototyping
- keine allgemein anerkannte Methodik für prototypingorientierte Software-Entwicklung
- keine ausgereifte Werkzeugtechnologie (einige vielversprechende Ansätze)

Einigkeit besteht darüber, dass

- Prototypen schnell und billig hergestellt werden müssen
- Prototypen wichtige Bestandteile der Software-Entwicklung im Hinblick auf die Qualitätssicherung und Risikominderung sind
- Prototypen wichtige Lernhilfen sind, um den SoftwareEntwicklungsprozess zu verbessern
- ökonomisches Prototyping nicht ohne Werkzeuge möglich ist

Prototyping

SOFTWARE ENGINEERING

Idee

Möglichst schnell ein lauffähiges Modell haben

» das das gewünschte System simuliert

» mit dem man experimentieren kann

» mit dem der Auftraggeber prüfen kann, ob er richtig verstanden worden ist

Zweck

» Missverständnisse möglichst früh erkennen

» Spezifikation möglichst früh vervollständigen

» verschiedene Lösungsansätze ausprobieren

Prototyping: Begriffe und Abgrenzung

SOFTWARE ENGINEERING

Ein Software-Prototyp ist ein – mit wesentlich geringerem Aufwand als das geplante Produkt hergestelltes – einfach zu änderndes und zu erweiterndes ausführbares Modell des geplanten Software-Produkts oder eines Teiles davon, das nicht notwendigerweise alle Eigenschaften des Zielsystems aufweisen muss, jedoch so geartet ist, dass vor der eigentlichen Systemimplementierung der Anwender die wesentlichen Systemeigenschaften erproben kann.

Prototyping umfasst alle Tätigkeiten, die zur Herstellung solcher Prototypen notwendig sind.

Prototyping: Begriffe und Abgrenzung

SOFTWARE ENGINEERING

Prototyping verfolgt verschiedene Ziele. Dies hat zur Folge, dass sowohl zwischen

- verschiedenen *Arten des Prototyping*
- als auch verschiedenen *Arten von Prototypen*

unterschieden werden muss.

Arten des Prototyping

SOFTWARE ENGINEERING

» *Exploratives* Prototyping

Funktionserforschung durch Prototyping führt zur *Spezifikation*

» *Experimentelles* Prototyping

Simulation
- der Benutzerschnittstelle
- der Systemfunktionen
- der Systemarchitektur

führt zur *Spezifikation* und *Systemarchitektur*

» *Evolutionäres* Prototyping

Beginnt mit einer Teilmenge der Funktionen
Schrittweise Hinzunahme neuer Funktionen

Exploratives Prototyping

SOFTWARE ENGINEERING

Ziel

möglichst vollständige ***Systemspezifikation***

Zweck

- Einblick der Entwickler in Anwendungsbereich
- verschiedene Lösungsansätze frühzeitig mit Anwender diskutieren
- Realisierbarkeit im gegebenen Umfeld frühzeitig abklären

Vorgangsweise

- Prototyp (mindestens der Benutzerschnittstelle) entwickeln
- Prüfung anhand realer Anwendungsfälle, sukzessive erwünschte Funktionalität ausloten
- Prototyperweiterung

Entwicklungsmannschaft

Anwender und Entwickler

Qualitätsaspekte

Maßgeblich ist nicht die Qualität der Konstruktion des Prototyps, sondern die Funktionalität, leichte Veränderbarkeit und Kürze der Entwicklungszeit

Prototyp-Art

unvollständiger Wegwerf- oder wiederverwendbarer Prototyp

Experimentelles Prototyping

SOFTWARE ENGINEERING

Ziel

vollständige ***Spezifikation der software-technischen Aspekte*** der Systemarchitektur

Zweck

Tauglichkeit von "Objekt"-Spezifikationen, Architekturentscheidungen und Lösungsideen für Systemkomponenten vor der eigentlichen Implementierung nachzuweisen

Vorgangsweise

- Systemdesign erstellen
- Prototyp für Systemarchitektur entwickeln
- Prüfung der Adäquatheit und Erweiterbarkeit der Systemarchitektur anhand des Prototyps und realer Anwendungsfälle
- Prototypen für komplexe Komponenten entwickeln

Entwicklungsmannschaft

Softwaretechniker

Qualitätsaspekte

wie bei explorativem Prototyping

Prototyp-Art

(un)vollständiger Wegwerfprototyp

Evolutionäres Prototyping

SOFTWARE ENGINEERING

Ziel

Meisterung der Komplexität durch Steigerung des Qualitätsmerkmals Erweiterbarkeit

Vorgangsweise

Inkrementelle Systementwicklung (Prototyp für von vornherein klare Benutzeranforderungen --> Basissystem für Anwender und nächsten Entwicklungsschritt --> sukzessive Hinzunahme neuer Eigenschaften aufgrund der bisherigen Systemanwendung)

Entwicklungsmannschaft

Softwaretechniker

Qualitätsaspekte

Qualität der Konstruktion des Prototyps muss der allgemeinen softwaretechnischen Qualitätsanforderungen entsprechen

Prototyp-Art

wiederverwendbarer Prototyp

Arten von Prototypen

SOFTWARE ENGINEERING

- vollständige und unvollständige Prototypen
- Wegwerfprototypen und wiederverwendbare Prototypen
- horizontale und vertikale Prototypen
- Demonstrationsprototypen
- Labormuster
- Pilotsysteme

Vollständiger Prototyp

SOFTWARE ENGINEERING

» ein Prototyp im herkömmlichen Sinne

» ein Prototyp in dem alle wesentlichen Funktionen des geplanten Systems vollständig verfügbar sind

Die bei der Herstellung und beim Einsatz gemachten Erfahrungen und der Prototyp selbst bilden die Grundlage für die endgültige Systemspezifikation.

Vollständige Prototypen werden für Softwaresysteme kaum hergestellt.

Unvollständiger Prototyp

SOFTWARE ENGINEERING

Software, die es gestattet, die Brauchbarkeit und/oder Machbarkeit *einzelner Aspekte* wie

- **Benutzerschnittstelle,**
- **Systemarchitektur,**
- **Systemkomponenten**

des geplanten Systems zu überprüfen.

Wegwerfprototyp

SOFTWARE ENGINEERING

Von einem *Wegwerfprototypen* spricht man, wenn die Implementierung des Prototyps bei der Implementierung des Zielsystems nicht weiterverwendet wird, sondern der Prototyp „nur“ als *ablauffähiges Modell* dient.

Wiederverwendbarer Prototyp

SOFTWARE ENGINEERING

Von *wiederverwendbaren Prototypen* spricht man, wenn *wesentliche Teile des Prototyps* bei der Implementierung des Zielsystems übernommen werden können.

Horizontaler Prototyp

SOFTWARE ENGINEERING

Ein *horizontaler Prototyp* realisiert nur spezifische Aspekte eines Softwareproduktes (wie zum Beispiel die Benutzungsschnittstelle oder die Datenhaltung), ohne die dahinter liegende Funktionalität vollständig zur Verfügung zu stellen.

Vertikaler Prototyp

SOFTWARE ENGINEERING

Ein *vertikaler Prototyp* implementiert einzelne Komponenten des geplanten Produkts vollständig,

um zum Beispiel die Erfüllung von Performanceanforderungen oder die technische Machbarkeit experimentell abzuklären.

Demonstrationsprototyp

SOFTWARE ENGINEERING

Demonstrationsprototypen dienen als Akquisitionsinstrumente, um potenziellen Auftraggebern die wichtigsten Produkteigenschaften auf anschauliche Weise demonstrieren zu können.

Labormuster

SOFTWARE ENGINEERING

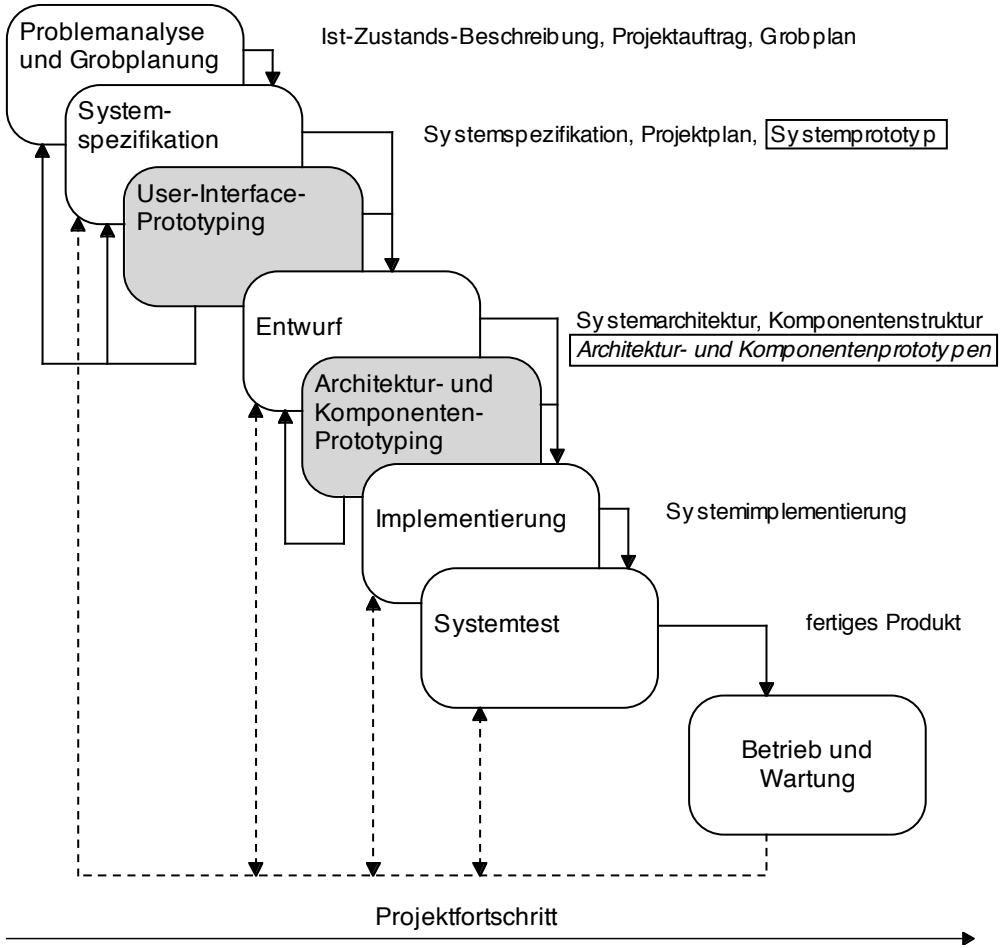
Labormuster werden verwendet, um konstruktionsbezogene Aspekte zu evaluieren und zu demonstrieren, zum Beispiel die Beschaffenheit und Tauglichkeit von Architekturmustern.

Pilotsystem

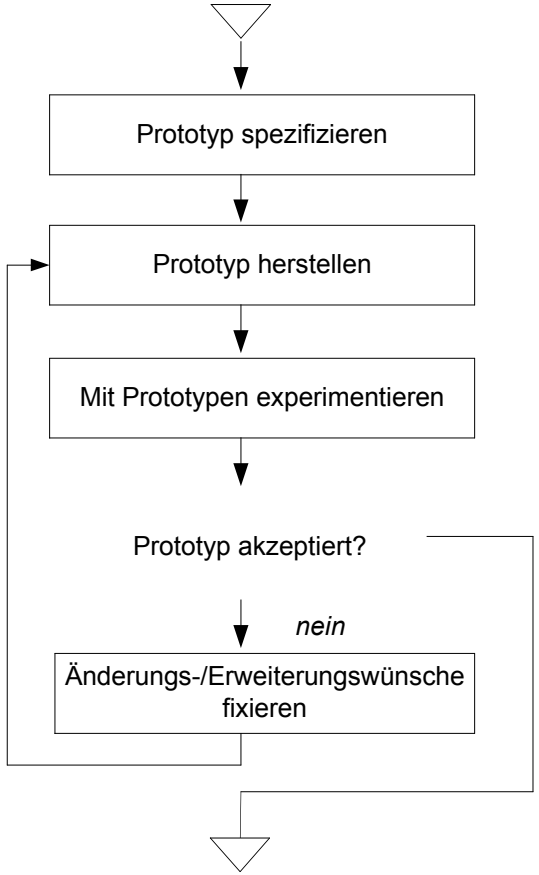
SOFTWARE ENGINEERING

Unter einem *Pilotsystem* verstehen wir ein Softwaresystem, das bereits einen gewissen Reifegrad (Funktionsumfang, Stabilität) erreicht hat und als Basis für eine Produktentwicklung herangezogen wird.

Prototyping-orientiertes Prozessmodell



Prototyping-Aktivitäten



Aufwand und Eignung von Prototyping

SOFTWARE ENGINEERING

Durch Prototyping ergibt sich

weniger Entwurfsaufwand	mehr Implementierungsaufwand
weniger Testaufwand	
weniger volle Iterationen	mehrere kurze Schleifen des Life-Cycles durch Rückkopplung

Eignung

Prototyping eignet sich gut für Systeme, für die es kein Vorbild gibt:

- zur Funktionserforschung
- zur Modellierung der Benutzerschnittstelle
- zur Architekturbewertung

Prototypingorientierte Software-Entwicklung

SOFTWARE ENGINEERING

Vorteile

- Stärken dort, wo life-cycle-orientierte Entwicklungsmethodik Schwächen aufweist
- die Vorteile der life-cycle-orientierten Entwicklungsmethodik bleiben erhalten
- projektbegleitende Qualitätssicherung wird gefördert und erleichtert
- führt i.d.R. zur Senkung der Produkt-Lebenszykluskosten

Nachteile

- bedingt mehr und spezialisiertere Werkzeuge als durch konventionelle Programmierumgebungen zur Verfügung gestellt

Weitere Bemerkungen siehe nächste Seiten

Einsatzbereiche für das Prototyping

SOFTWARE ENGINEERING

Gibt es Applikationsklassen, die besonders für prototypingorientierte Entwicklung geeignet sind und solche, die nicht dafür geeignet sind?

Weitverbreitete Ansicht

Prototyping ist eine nützliche Technik zur Spezifikation und Entwicklung von Applikationen aus dem kommerziellen Anwendungsbereich (z.B. für Informationssysteme), aber kaum einsetzbar im technisch-wissenschaftlichen Anwendungsbereich (z.B. für Prozessautomation, Real-Time-Anwendungen).

Unsere Ansicht

- Der (messbare) Nutzen prototypingorientierter Software-Entwicklung in verschiedenen Anwendungsbereichen ist unterschiedlich, die Ursache dafür ist jedoch meist das Fehlen entsprechender Werkzeuge und nicht die mangelnde Anwendbarkeit dieser Technik.
- Jede neue Applikation, für die die gewünschte *Funktionalität*, die *Art der Funktionsnutzung* oder die *Realisierbarkeit* bestimmter funktionaler Aspekte nicht zweifelsfrei feststeht, ist ein Kandidat für prototypingorientierte Software-Entwicklung.

  => In der Praxis gibt es kaum Softwareprojekte, auf die dies nicht zutrifft, insbesondere auch nicht im technischen Anwendungsbereich.

Prototyping und End-User-Computing

SOFTWARE ENGINEERING

In welchem Ausmaß sollen die Anwender in den Software-Entwicklungsprozess einbezogen werden?

Die Empfehlungen reichen von 50% Anwender- bis 50% Softwaretechniker-Anteil bis hin zum "End-User-Computing" (100 % Anwenderanteil - Softwaretechniker hat beratende Funktion)

Unsere Ansicht

- *End-User-Computing hat zweifellos einige Vorteile*
  - Missverständnisse infolge Kommunikationsschwierigkeiten zwischen Anwender und Entwickler entfallen
  - Akzeptanzprobleme entfallen (Personalunion von Entwickler und Anwender)
  - Koordinierungsprobleme bei Systemeinführung werden vermindert

Prototyping und End-User-Computing

SOFTWARE ENGINEERING

» *Nachteile des End-User-Computing*
- führt in der Regel zu Insellösungen
- verhindert Schwachstellenidentifikation und Innovation
- bildet hohes Risiko im Hinblick auf Integritätsanforderungen (Konsistenz, Vollständigkeit)
- führt meist zu redundanten Systemen
- die besten Softwaresysteme sind nicht von Anwendern entwickelt worden (Ergonomie, Effizienz, Eleganz, Integrität erfordern ein hohes Maß an Spezialwissen)

⇒ Von End-User-Computing ist (mit einigen Ausnahmen) abzuraten

» *Benutzerbeteiligung aktivitätsabhängig*
- Spezifikationsprototyping gemeinsame Verantwortlichkeit Benutzer/-Entwickler
  Aufwandsverteilung 60:40 (Validation ist aufwendiger als Implementierung)
- Architekturprototyping
  Verantwortlichkeit liegt ausschließlich beim Entwickler Benutzer nur Consulting-Funktion bei Experimenten (Bereitstellung realer Anwendungsbeispiele, Datenflussverifikation)
  Aufwandsverteilung Benutzer/Entwickler 20:80

Prototyping und inkrementelle Software-Entwicklung

SOFTWARE ENGINEERING

Kann Prototyping als evolutionäre Entwicklungsstrategie zur Generierung von Software-Systemen mit Produktcharakter eingesetzt werden?

- methodologisch ja, jedoch abhängig von der Qualität der Werkzeuge
- heute kaum Werkzeuge verfügbar, die die entsprechenden Anforderungen hinsichtlich Performance, Strukturiertheit, Sicherheit und Lesbarkeit der Implementierung erfüllen

Prototyping und Topdown-Strategie

SOFTWARE ENGINEERING

Wie unterscheidet sich eine prototyping- von einer topdown-orientierten Software-Entwicklung?

Topdown-Strategie

(1) Systemspezifikation
(2) Festlegen der Systemgrobstruktur (Datenflussdiagramme, Modulstruktur, ...)
(3) Implementierung eines Basissystems (der High-Level- oder Control-Module) in der Zielsprache
(4) Topdown-Test (Benutzung von "stubs", "dummy routines" für Low-Level-Module)
(5) Evaluation, (Re-Design), sukzessive Implementierung, …

Prototypingorientierte Software-Entwicklung ist top-down-orientierte Software-Entwicklung.

- » kein Wiederspruch zur Topdown-Strategie
- » Unterschied und Vorteil der pto-SE zur tdo-SE
  - – Basissystem (initial prototyp) wird entwickelt ohne Code in Zielsprache zu schreiben
  - – verwendete Werkzeuge (UI-Generatoren, Hypertext, 4GL, Web-Tools, ...) weisen höheren Abstraktionsgrad auf
  - – Benutzer kann stärker in Topdown-Test einbezogen werden

=> realitätsnahe Funktionstests schneller möglich

Kosten-, Risiko- und Qualitätsaspekte

SOFTWARE ENGINEERING

Prototypingorientierte Software-Entwicklung bringt

- keine signifikante Verminderung der Entwicklungskosten u. -zeit, teilweise sogar das Gegenteil
- eine signifikante Qualitätssteigerung der Qualitätsfaktoren
  - funktionale Adäquanz
  - Benutzerfreundlichkeit
  - Änderbarkeit und Erweiterbarkeit
  - Korrektheit und Zuverlässigkeit
    (äußerst effiziente Qualitätssicherungsmaßnahme)
- eine drastische Senkung der SLC-Kosten durch Verminderung der Wartungs- und Betriebskosten infolge der Qualitätssteigerung
- eine signifikante Verminderung des Testaufwandes für die Implementierung

  => Erhöhung der Entwicklungsproduktivität

Kosten-, Risiko- und Qualitätsaspekte

SOFTWARE ENGINEERING

- eine signifikante Verbesserung der Planbarkeit (Termintreue) von Software-Projekten
  - verbesserte Systemspezifikation und Verifikation der Architektur ermöglicht genauere Abschätzung des Implementierungs- und Testaufwands
  - Methodik unterstützt den Prozess der arbeitsteiligen Software-Entwicklung besonders gut (Verringerung von Kommunikationsschwierigkeiten, verbesserte Teilespezifikation, sicherere Zerlegungsstruktur)

- eine Reduktion des "Flop-Risikos”
  - experimentell abgesicherte Modellverifikation und Durchführbarkeitsstudie vermindert die Anzahl unsicherer Annahmen

Risiken in Verbindung mit Prototyping

SOFTWARE ENGINEERING

Prototyping führt einerseits zu einer beträchtlichen Verminderung der Risikofaktoren im Software-Entwicklungsprozess, birgt jedoch andererseits neue Risikofaktoren in sich.

- Nichtkonvergenter Spezifikationsprozess
  "the more they get, the more they want“
- Überspezifikation – Gefahr der Vernachlässigung von Eleganz und Effizienz einer Lösung zugunsten verzichtbarer Flexibilität
- Vernachlässigung softwaretechnischer Prinzipien in der Systemimplementierung (Gefahr des Verlusts von Qualitätsmerkmalen, allen voran Struktur und Erweiterbarkeit) Gefahr der Auslieferung von Systemprototypen statt Systemprodukten
- Schwierigkeiten bei der Kostenrechtfertigung

Prototyping und Trends im Software Engineering

SOFTWARE ENGINEERING

Erfahrungen mit prototypingorientierter Software-Entwicklung beweisen die Qualitätssteigerungs- und Kostenverminderungseffekte.

Trends

- Das prototypingorientierte Vorgehensmodell hat in weiten Bereichen bereits die konventionellen Phasenmodelle abgelöst.
- Laufende Verbesserungen von existierenden Prototypingwerkzeugen und umfangreiche Neuentwicklungen durch Marktdruck.
  (Der Aufwand, den Software-Lieferanten in solche Entwicklungen stecken, ist beträchtlich.)
- Viele neue Entwicklungsumgebungen unterstützen Prototypingaktivitäten.
- Die Anzahl der Unternehmen, in denen diese Methodik eingesetzt wird, nimmt kontinuierlich zu.

Prototyping und Werkzeuge

SOFTWARE ENGINEERING

Sind spezielle Werkzeuge notwendig für prototypingorientierte Software-Entwicklung?

» Um unsere Auffassung von Prototypen zu realisieren - ja!

um schnell, einfach zu modifizierende lauffähige Systemmodelle zu erzeugen sind Software-*Werkzeuge* unabdingbar

"Creating and changing a prototype is quick, easy and painless – if it is anything else, the wrong tools are being employed".

J.L. Connel, L.B. Shafer

Das Spiralmodell

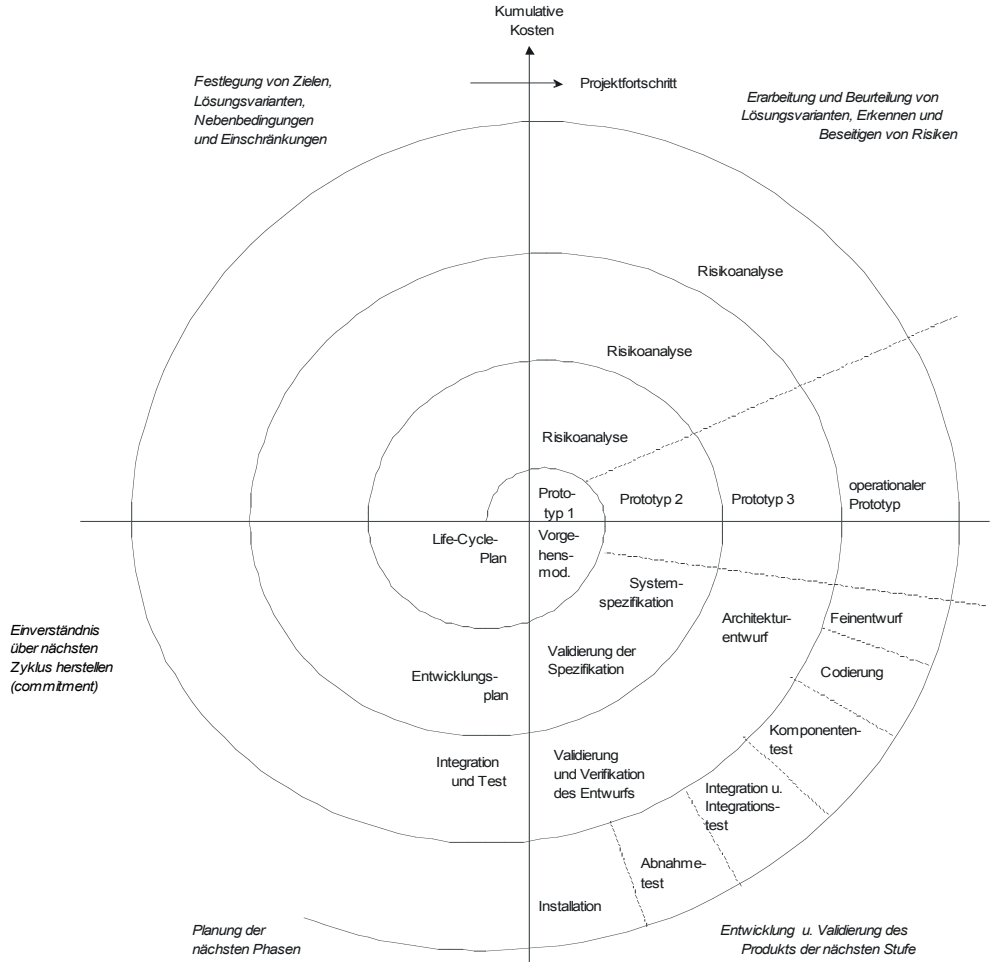
SOFTWARE ENGINEERING

Boehm (1988) hat ein Prozessmodell vorgeschlagen, das die bisher vorgestellten Modelle kombiniert bzw. es gestattet, sie in dieses Modell einzubetten.

Beim Spiralmodell handelt es sich daher um ein Metamodell, das die Möglichkeit bietet, die für ein Software-Entwicklungsprojekt am besten geeignete Vorgehensweise zu wählen.

Das Spiralmodell von Boehm

Der Entwicklungsprozess wird nach Boehm in vier Schritte gegliedert, die im Sinne einer evolutionären Entwicklungsstrategie mehrmals zyklisch durchlaufen werden



Das Spiralmodell von Pomberger/Pree

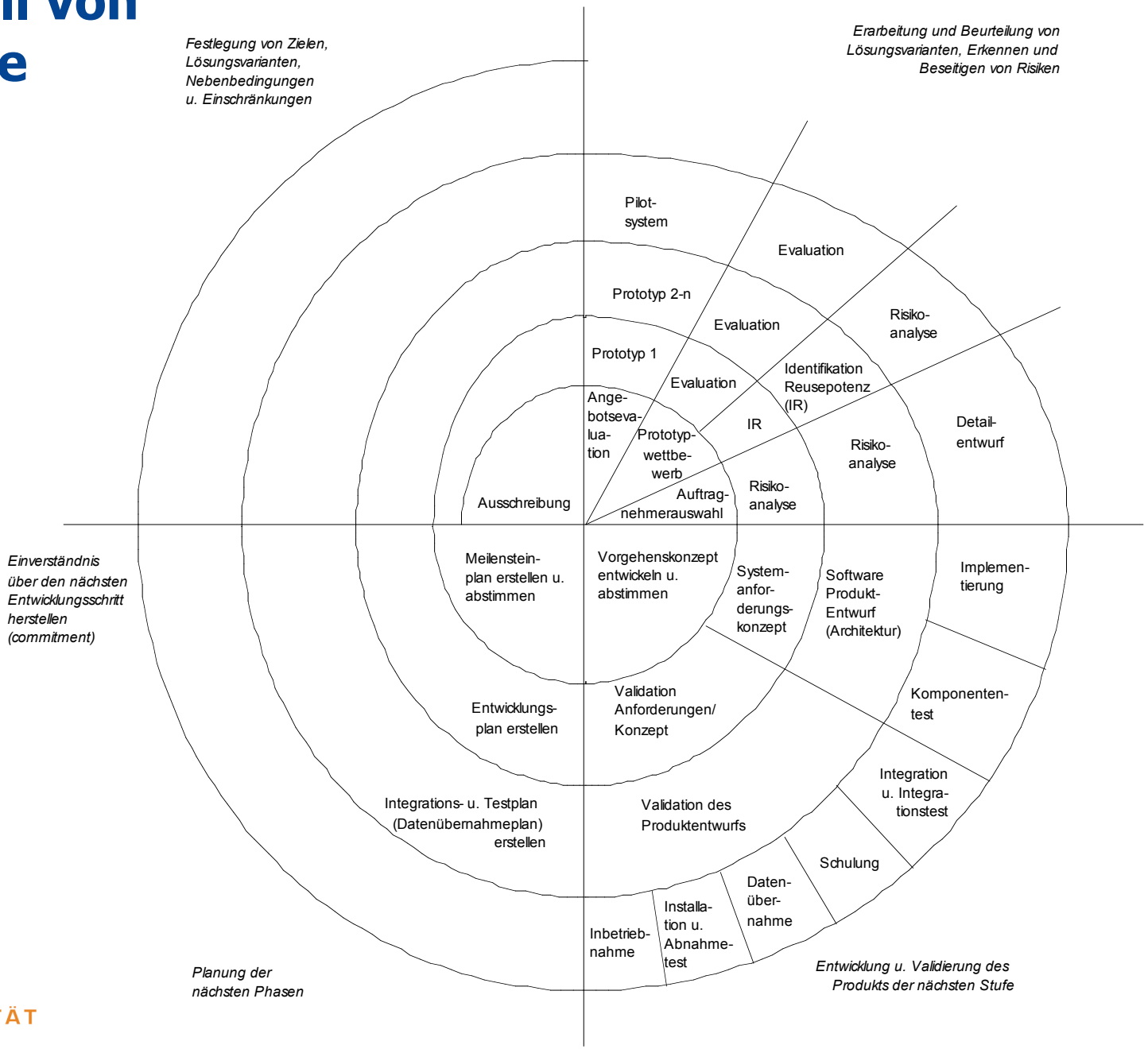
SOFTWARE ENGINEERING

Den in den vorhergehenden Abschnitten behandelten Prozessmodellen ist gemeinsam, dass sie nur auf den Software-Entwicklungsprozess ausgerichtet sind.

Wenn Software benötigt, dann ist es sinnvoll, zu entscheiden, ob die benötigte Software überhaupt entwickelt werden muss, oder ob die Möglichkeit besteht sie (billiger) zuzukaufen.
Es ist eine „make or buy – Entscheidung“ erforderlich und es ist sinnvoll, die dazu erforderlichen Aktivitäten in das Prozessmodell zu integrieren.

Das Spriralmodell von Boehm weist die Schwäche auf, dass die Risikoanalyse der Prototypentwicklung vorgelagert ist, aber gerade im Zuge der Protypentwicklung können signifikante Risikofaktoren identifiziert werden.

Das Spiralmodell von Pomberger/Pree



Unterschied zum Spiralmodell von Boehm

SOFTWARE ENGINEERING

- Evaluierung von Alternativen erst nach den Prototyping-Aktivitäten

  ➔ Evaluationsprozess sicherer und effizienter

- Einführung einer (in der Boehm´schen Modellvariante nicht vorgesehenen) Aktivität *Identfikation von Wiederverwendungspotenzialen*

  ➔ Förderung wiederverwendungsorientierter Produktentwicklung

- Integration des „make-or-buy-“ Entscheidungsprozesses

- Verlagerung der Risikoanalyse (letzte Aktivität im Schritt 2 und nicht als erste wie bei Boehm)

  ➔ präzisere Risikoszenarien sind möglich

Vorteile

SOFTWARE ENGINEERING

- Prozessmodell ist umfassender als die meisten anderen Prozessmodelle
- Prozessmodell ist ein Metamodell und gestattet die Integration anderer Prozessmodelle als Spezialfälle

  ➔ höhere Flexibilität, bessere Anpassungsmöglichkeiten an spezielle Rahmenbedingungen, höhere Akzeptanz
- Reduktion des Floprisikos
- Fehler und ungeeignete Lösungsansätze (im technischen Bereich, in der Personalzusammensetzung, in der Ausprägung des Qualitätssicherungsprozesses) werden früh erkannt

  ➔ Organisation des Entwicklungsprozesses kann schnell und zielgerichtet umdirigiert werden
  ➔ Vermeidung unnötiger Kosten und Steigerung der Produktqualität weil

Nachteile

SOFTWARE ENGINEERING

- Richtiges „Tailoring“ hängt in entscheidendem Ausmaß von den Fähigkeiten und Erfahrungen der Projektverantwortlichen ab (Metamodellproblematik)
- Akzeptanzprobleme durch Prototyping-Wettbewerb
- Abschlagszahlungen im Zuge des Prototyping-Wettbewerb werden manchmal als verlorene Zusatzkosten gesehen

Fallstudienergebnisse zum Spiral-Modell

SOFTWARE ENGINEERING

Ausschreibung / Angebots- und Anbieterevaluation

	Konventionelles Vorgehensmodell	Prototyping-orientiertes Spiral-Modell
Aufwand Ausschrei-bungsdokumentation	110 PT	24 PT
Lastenheftumfang	64 Seiten	17 Seiten
Zeit*dauer* Angobts-/Anbieter-Evaluation	unbekannt	4 Wochen
Zeit*aufwand* Angebots-/Anbieter-Evaluation	unbekannt	15 PT (12i/3e)
Prototypingaufwand Anbieter	-	40/55 PT

Fallstudienergebnisse zum Spiral-Modell

SOFTWARE ENGINEERING

Herstellungsprozess

	Konventionelles Vorgehensmodell	Prototyping - orientiertes Spiral -Modell
Pflichtenheft	298 Seiten	-
PT-Wiederverwen - dungsgrad	-	90% (36 PT)
Herstellungsdauer	> 250% (35 Monate)	100% (14 Monate)
Herstellungsaufwand	geschätzt 85 PM	25 PM
Herstellungskos ten auftraggeberseitig	210% bis Abbruch	100%
Benutzerakzeptanz	1	5,5
Kosten externe Koordination	-	4% der Herstellungsk.

Der Unified Process

SOFTWARE ENGINEERING

Das Prozessmodell „Unified Process“ wurde von der Firma Rational entwickelt (das Modell wird deshalb häufig als „Rational Unified Process“ oder kurz RUP bezeichnet) und baut auf Überlegungen von Ivar Jacobson zur Systematisierung des Software-Entwicklungs-prozesses auf.

RUP (Rational) Unified Process

SOFTWARE ENGINEERING

Das Prozessmodell postuliert im Wesentlichen

- ***sechs grundlegende Entwicklungsschritte:***
  Geschäftsprozessmodellierung (*Business Modeling*), Anforderungsanalyse (*Requirements*), Analyse und Entwurf (*Analysis & Design*), Implementierung (*Implementation*), Test und Installation (*Deployment*)

- ***drei Unterstützungsprozesse:***
  Konfigurations- und Änderungsmanagement (*Configuration & Change Management*), Projektmanagement (*Project Management*) und Umgebungsmanagement (*Environment Management*)

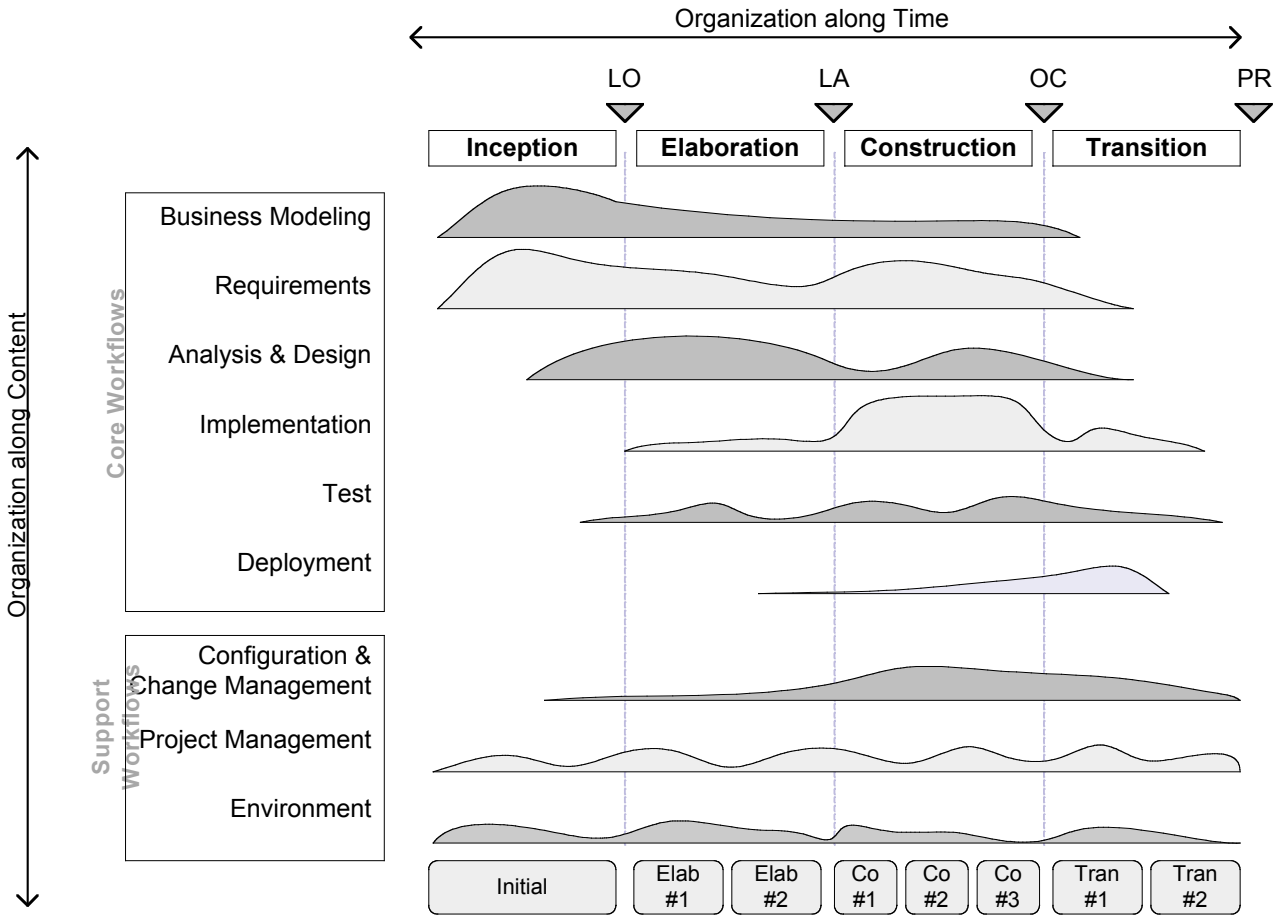
- und ***vier Phasen:***
  Anfangsphase (*Interception*), Ausarbeitungsphase (*Elaboraion*), Konstruktionsphase (*Construction*) und Übergangssphase (*Transition*)

RUP – charakteristische Eigenschaften

SOFTWARE ENGINEERING

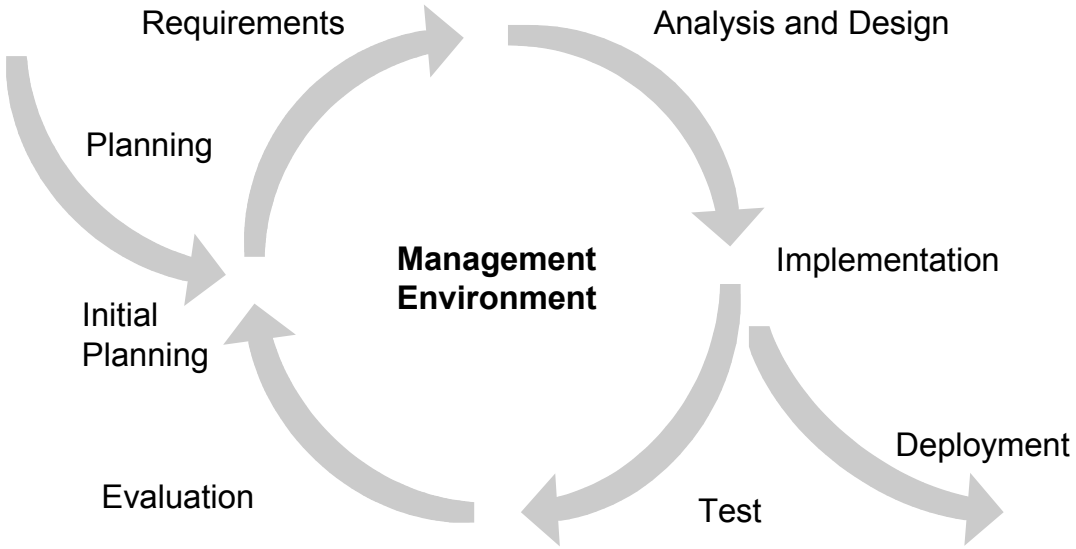
- Das Prozessmodell ist anwendungsfall- und architekturzentriert
- Das Modell postuliert sechs Entwicklungsschritte, drei Unterstützungsprozesse, vier Phasen
- Jede Phase endet mit einem Meilenstein
- Der Unified Process ist ein iterativer und inkrementeller Prozess
- Nach anfänglichen Planungsschritten wird jeder Entwicklungsschritt iterativ mehrmals durchgeführt
- Im Zuge der wiederholten Ausführung der Entwicklungsschritte werden Umfang und Qualität der Arbeitsergebnisse (Produkte) schrittweise und kontinuierlich verbessert
- Im Zuge einer eines Entwicklungszyklus (Iteration), werden so viele Produkte wie möglich bzw. notwendig weiterentwickelt
- Wenn alle gewünschten Produkte des geplanten Softwaresystems in der erforderlichen Qualität vorliegen, endet der Iterationsprozess und das Softwaresystem kann ausgeliefert bzw. installiert werden
- Die Abfolge der Iterationen ist in Phasen gegliedert und in jeder Phase können eine oder mehrere Iterationen durchlaufen werden
- Die Anzahl erforderlicher Iterationen variiert von Projekt zu Projekt und ist abhängig von dessen Komplexität
- Die Anforderungen und Leistungsmerkmale an das Produkt wird in Form von Anwendungsfällen (*Use Cases*) beschrieben
- Der Entwicklungsprozess wird quasi von den Anwendungsfällen gesteuert

RUP Übersicht



Quelle: Kruchten, 2000

RUP: Iterativer und inkrementeller Prozess



Quelle: Kruchten, 2000

RUP: Iterationsanzahl

SOFTWARE ENGINEERING

Projekt - Komplexität	Summe der Iterationen	Iterationen Interception	Iterationen Elaboration	Iterationen Construction	Iterationen Transition
niedrig	3	0	1	1	1
Normal	6	1	2	2	1
hoch	9	1	3	3	2

Quelle: Kruchten, 2000

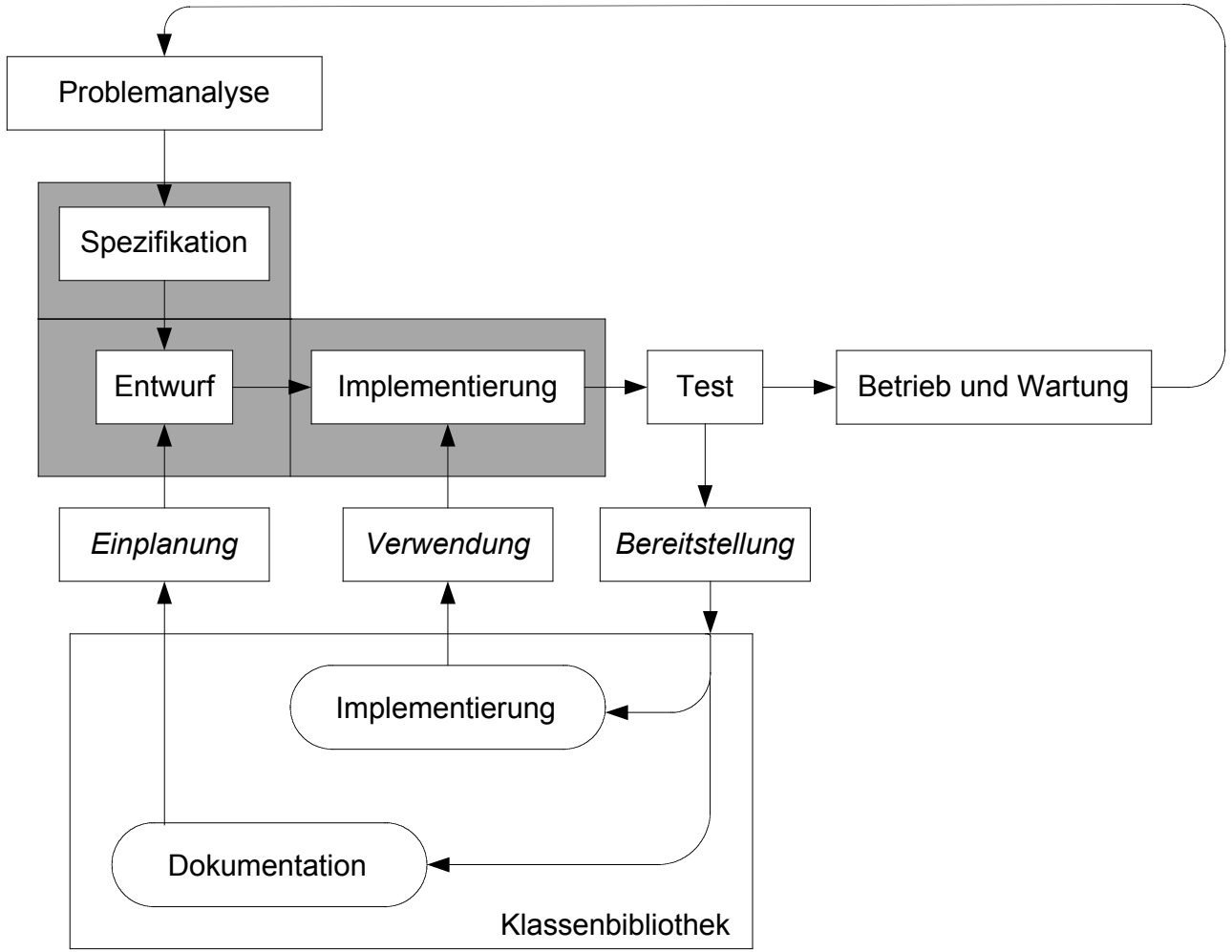
OO Phasenmodell - Ausgangssituation

SOFTWARE ENGINEERING

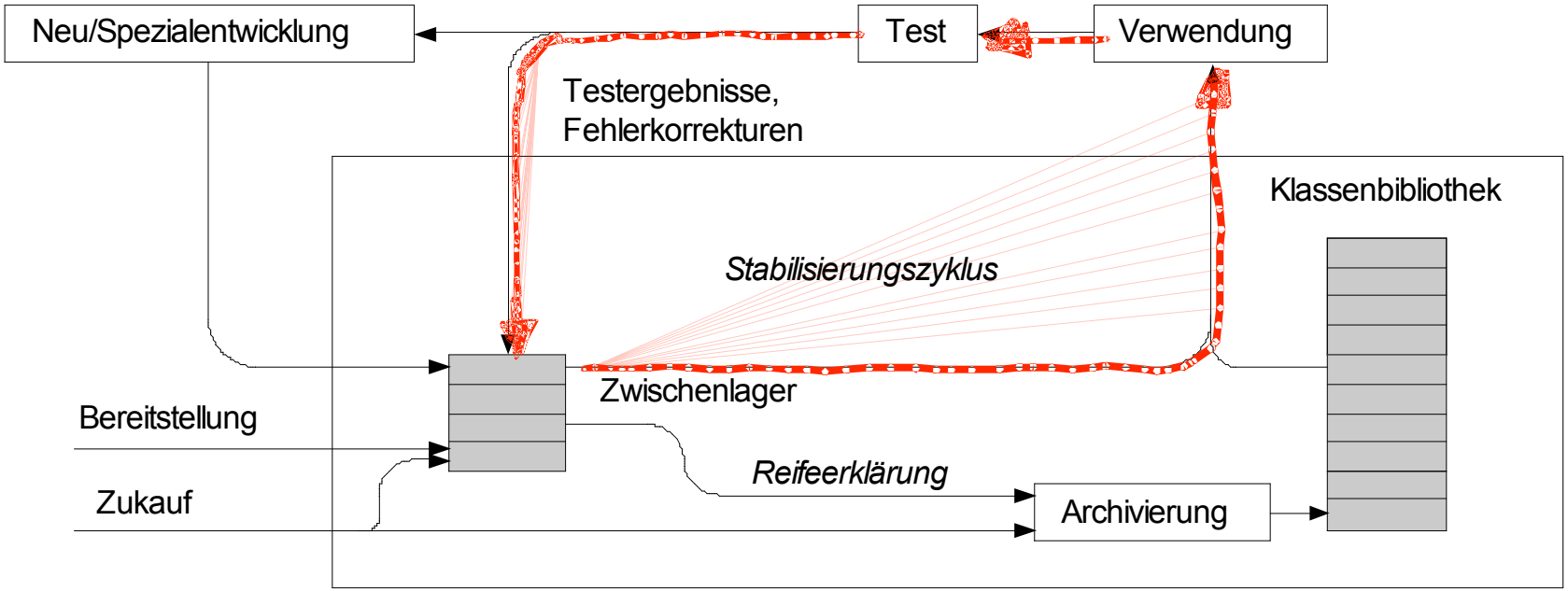
Ein wesentliches Merkmal der objektorientierten Software-Entwicklung besteht darin, dass die Wiederverwendung von (Objekt-)Klassen, von Klassenbibliotheken und von Frameworks durch Komposition, Vererbung und Polymorphie unterstützt wird.

Die vielfältigen Aspekte der Wiederverwendung – es können eigen- oder fremdentwickelte Architekturmodelle, Frameworks, Subsysteme, Klassenbibliotheken, Klassen, etc. wieder verwendet werden – müssen im Prozessmodell berücksichtigt werden.

OO Phasenmodell – Aufbau



Detailmodell Klassenbibliotheks-Administration



Vor- und Nachteile

SOFTWARE ENGINEERING

Vorteile

- speziell auf eine bestimmte Design- und Implementierungstechnologie zugeschnitten
- Konzentration auf Wiederverwendung von Komponenten

Nachteile

- Implementierungstechnologie explizit vorgegeben

Leichgewichtige (agile) Prozessmodelle

SOFTWARE ENGINEERING

Software-Entwicklungsprozessmodelle werden üblicherweise in zwei Gruppen unterteilt:

- » schwergewichtige
- » leichtgewichtige

Unter „schwer“ bzw. „leicht“ versteht man den Grad der Formalisierung des Prozesses und der mit diesem verbundenen (Zwischen-)Ergebnisse bzw. (Zwischen-)Produkte.

Schwergewichtige Prozessmodelle sind demnach die streng phasenorientierten Modelle, wie zum Beispiel das klassische sequenzielle Prozessmodell, das Wasserfallmodell, das
V-Modell,und der Rational Unified Process.

Leichtgewichtige Prozessmodelle, auch *agile* Prozessmodelle genannt, sind flexible, schwach formalisierte, iterative Prozessmodelle wie zum Beispiel eXtreme Programming (XP) oder Scrum.

Die Spiralmodelle stellen gemäß dieser Klassifizierung einen Mittelweg dar.

Leichtgewichtige Prozessmodelle

SOFTWARE ENGINEERING

*X*P – Extreme Programming

postuliert 4 kritische Erfolgsfaktoren:

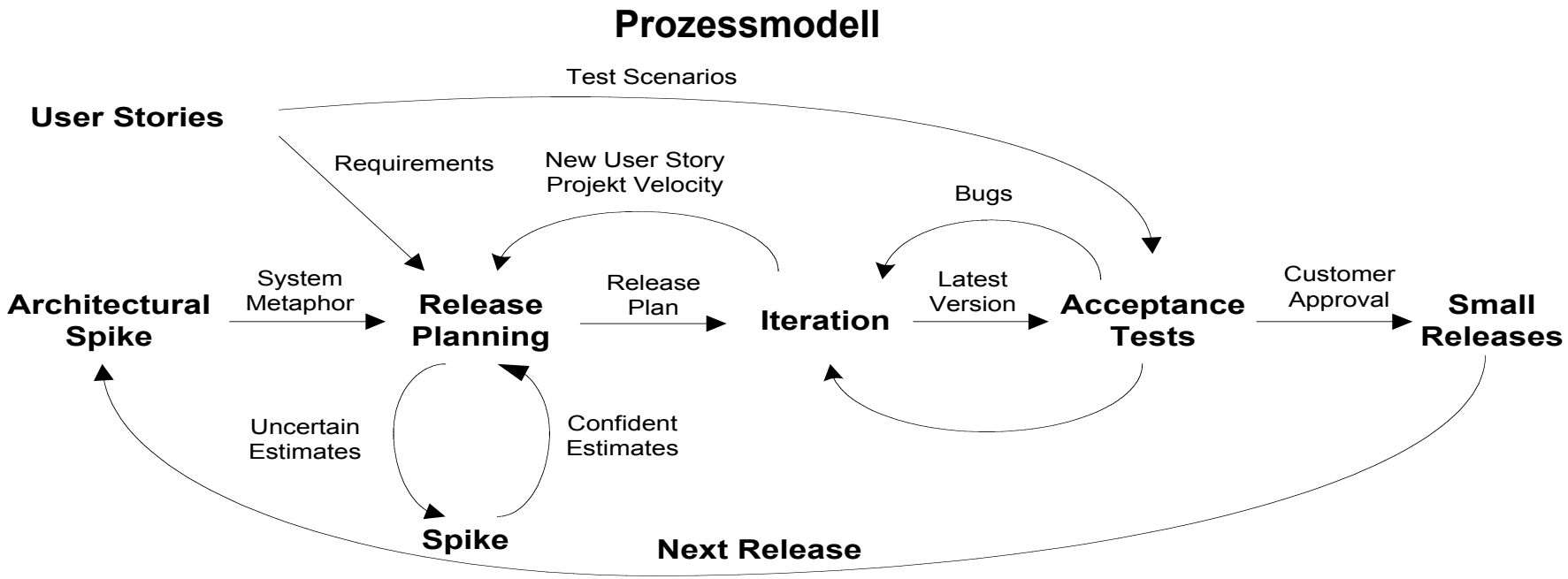
- *Kommunikation*
- *Einfachheit*
- *Feedback*
- *Courage*

Der ***Kommunikationsaspekt*** soll deutlich machen, dass der Projekterfolg wesentlich vom Ausmaß und der richtigen Art und Weise der Kommunikation zwischen den Projekt-beteiligten abhängt. Dem Schaffen von geeigneten Rahmenbedingungen zur Förderung der Kommunikation kommt daher besondere Bedeutung zu.

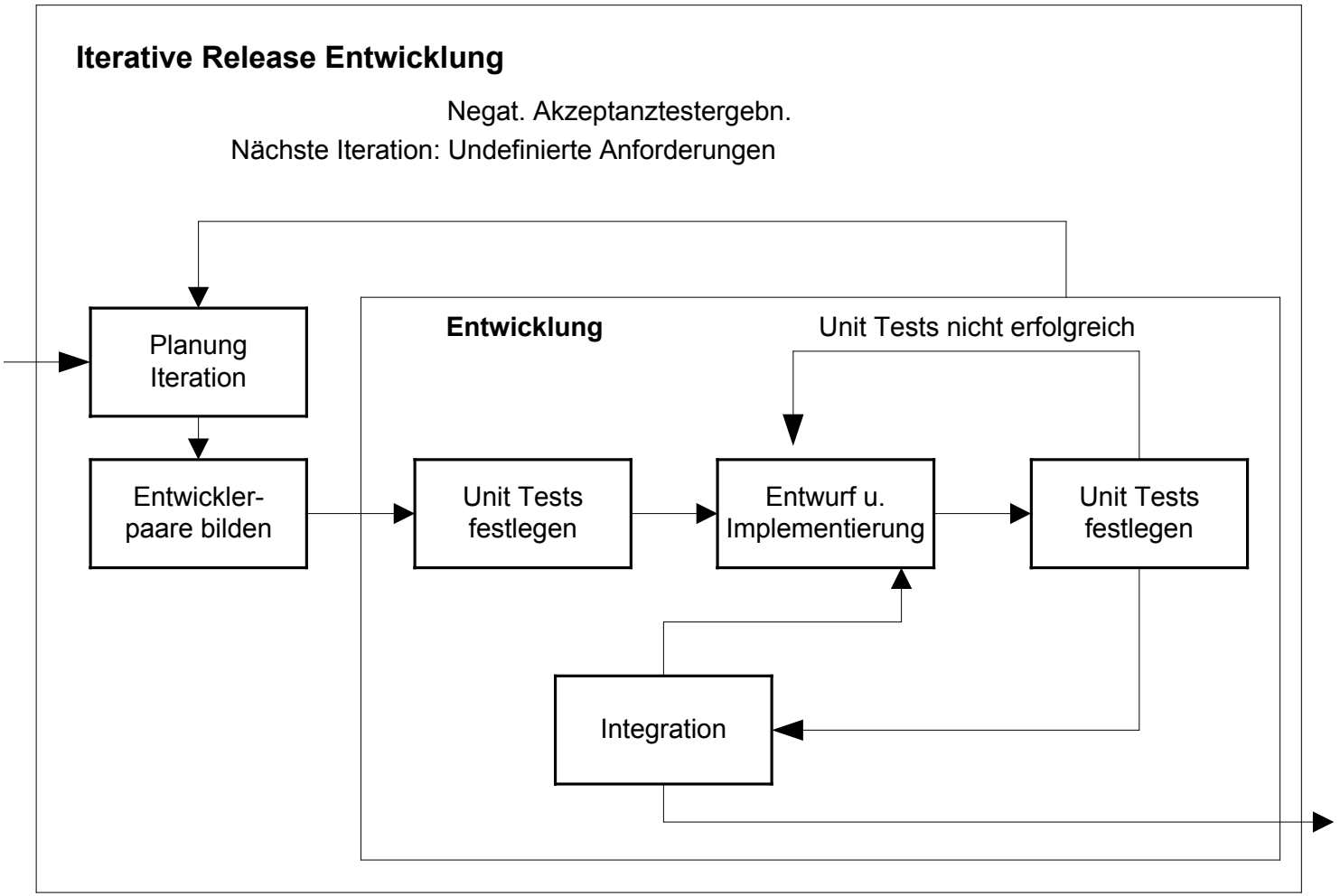
Der ***Einfachheitsaspekt*** postuliert dass man bei jedem Schritt und bei jedem Zwischenergebnis darauf achten muss, dass diese so einfach wie möglich gehalten werden.

Der ***Feedbackaspektes*** soll sicherstellen, dass die Entwickler laufend über den Grad der Zufriedenheit der Auftraggeber informiert werden, den Grad der Anwenderakzeptanz für jede ausgelieferte Komponente genau kennen und dass der Auftraggeber bei allen anstehenden Entscheidungen die daraus resultierenden Folgen im Hinblick auf Kosten, Termine, Produkteigenschaften und Produktqualität abzuschätzen imstande ist.

Durch die Betonung des ***Courageaspektes*** sollen die Projektbeteiligten zu verantwortlicher Eigeninitiative und dazu ermutigt werden, ohne pompöse Planung und hundertprozentige Absicherung ihrer Meinung nach für den Projekterfolg notwendige Schritte in Angriff zu nehmen und über entsprechende Kommunikation transparent machen.



Quelle: Donovan Wells


Iterative Release Entwicklung
Negat. Akzeptanztestergebn.
Nächste Iteration: Undefinierte Anforderungen
Planung Iteration
Entwicklung
Unit Tests nicht erfolgreich
Entwickler-paare bilden
Unit Tests festlegen
Entwurf u. Implementierung
Unit Tests festlegen
Integration

Zwölf Merkmale, respektive Praktiken, charakterisieren das XP-Entwicklungsparadigma:

(1) ***planing game, user stories***
Ziele, Leistungsmerkmale und der Funktionsumfang der einzelnen Releases werden im Rahmen eines Planungsspieles schnell und auf einfache Weise von den Auftraggebern gemeinsam Form von Story Cards dokumentiert.

(2) ***small releases***
Die Vorgehensweise der Systementwicklung ist releaseorientiert, d.h. in kurzen Abständen werden Systemreleases mit überschaubaren Funktionszuwachs entwickelt und ausgeliefert.

(3) Eine möglichst einfache System-Metapher bildet die Ausgangsbasis für den Entwicklungsprozess; jede unnötige Komplexität soll von vornherein vermieden werden.

(4) Entwurf und Implementierung jeder neuen Release muss mit besonderem Augenmerk nach dem Prinzip „so einfach wie möglich“ vorgenommen werden und unnötige (nicht zweifelsfrei erforderliche) Komplexität muss, sobald sie identifiziert ist, beseitigt werden.

(5) Kontinuierliches Testen gehört zu den Hauptaufgaben der Entwickler und der Anwender. Jeder Entwicklungsschritt beginnt mit der Festlegung von Unit Tests und endet erst nach erfolgreicher Durchführung aller spezifizierten Unit Tests. Die Anwender definieren Szenarien für System- und Akzeptanztests und setzen diese vor der Freigabe einer neuen Systemrelease um.

(6) ***refactoring***
Die Entwickler leisten Gewähr, dass entworfene und implementierte Komponenten kontinuierlich restrukturiert werden, um unnötige Komplexität zu beseitigen, den Code zu vereinfachen und flexibler zu gestalten, ohne die Funktionalität zu ändern.

(7) ***pair programming***
Die Entwickler arbeiten in Paaren. Jede Codezeile wird von zwei Entwicklern gemeinsam entwickelt und verantwortet.

(8) ***collective ownership***
Der gesamte Code ist gemeinsames Eigentum aller Entwickler. Jeder Entwickler kann (gemeinsam mit einem anderen) jede Codezeile zu jedem Zeitpunkt ändern.

(9) Der Integrationsprozess ist ein kontinuierlicher und kein punktueller Prozess. Ein Integrationsschritt findet statt, wenn eine Aufgabenerledigung abgeschlossen bzw. eine Komponente fertig gestellt ist, d. h. alle dafür spezifizierten Unit Tests erfolgreich durchgeführt wurden.

(10) Die Entwickler arbeiten nicht mehr als vierzig Stunden pro Woche, auch wenn es Terminprobleme gibt und wenn in einer Woche ausnahmsweise die vierzig Stunden Wochenarbeitszeit überschritten worden ist, dann darf dies die folgende Woche keinesfalls mehr geschehen.

(11) ***on-site customer***
Dem Entwicklungsteam gehören stets einer oder mehrere Auftraggebermitarbeiter an, die für die Softwareentwickler stets verfügbar sind, um mit ihnen Problemstellungen diskutieren bzw. offene Fragen an sie stellen zu können.

(12) Zur Förderung der Kommunikation und Steigerung der Effizienz werden die Programme nach einheitlichen Richtlinien/Programmierstandards, die sich das Team selbst gibt, entwickelt.

Vorteile

- Flexibilität bei der Anpassung an sich ändernde Rahmenbedingungen
- Beschränkung auf das Wesentliche
- iteratives Grundmodell
- starken Einbindung des Auftraggebers (on-site customer Prinzip),
- Verzicht darauf, bereits bei der Produktentwicklung künftige Anwenderanforderungen vorauszudenken

Nachteile

- postulierte Prozessorganisation nur für „kleine“ Projekte gut geeignet ist
- stark verteilte Produktentwicklung wird durch dieses Prozessmodell nicht unterstützt
- Aspekt der Wiederverwendung von Komponenten wird zu wenig Beachtung geschenkt.
- Verzicht auf eine Systemdokumentation kann bei langlebigen Produkten schwerwiegende Folgen haben.