Software Engineering

Informatik II.

2. Software-Entwicklung

– Der Software-Prozess –

Dipl.-Inform. Hartmut Petters

Vorwort – was ich noch zu sagen hätte ...

Basis dieser Vorlesung sind vor allem die folgenden Ausarbeitungen

- Vorlesungsskript „Software Engineering“
  von Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/ses/
- Vorlesungsskript „Informatik II – Software Engineering“
  von Frau Prof. Dr. Kühn FH Karlsruhe FB W
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Das Buch „Software Engineering“ 6. Auflage/2001
  von Prof. Ian Sommerville University of Lancaster (UK)
  Addison Wesley ISBN 3-8273-7001-9

Konkret entnommene Beiträge sind i.d.R. mit einem Quellen-Verweis gekennzeichnet – sollte dieser fehlen bitte ich um Nachsicht.

Den „roten Faden“ durch die Vorlesung habe ich dem Skript der Vorlesung von Prof. Dr. Martin Glinz entnommen und um eigene Beiträge erweitert bzw. aus den beiden anderen Quellen ergänzt.

Für die Möglichkeit der Verwendung der wesentlichen Inhalte möchte ich mich an dieser Stelle bei den Autoren herzlich bedanken.

Unterlagen – Software Engineering

Die Unterlagen zur Vorlesung finden Sie als Acrobat-Reader-PDF-Datei unter

http://hpetters.gmxhome.de

z.Zt. nur Folien.

Später:

- Notizen
- Aufgabenbeispiele

Was macht „gute“ Software aus?

- Wartbarkeit
  - Weiterentwicklung
  - Portierung
- Zuverlässigkeit
  - Betriebssicherheit
  - Abgesicherte Funktionalität
- Effizienz
  - Nutzung der Systemressourcen
  - Reaktions- und Verarbeitungszeit
- Benutzerfreundlichkeit
  - Selbsterklärend, intuitiv bedienbar
  - Dokumentation + Hilfe-Funktion

Software-Prozess / Prozess-Arten

- Software-Prozesse beschreiben den Ablauf der Entwicklung + Pflege von Software
- Prozess
  - Folge von Schritten, die zur Erreichung eines gegebenen Zwecks ausgeführt wird
  - Unterscheidung
    - Ad-Hoc-Prozess
      - Spontan
      - Individuell
      - Ungeregelt
    - Systematischer Prozess
      - Geplant
      - gelenkt

Software-Entwicklung im Ad-Hoc-Prozess

- Entwicklungsprozess im „unprofessionellen Bereich“
- Vage Sachziele +keine Langfristplanung
- Keine Termin- und Kostenvorgaben
- Keine Spezifikationen, bruchstückhafte Entwürfe
- Test nur ansatzweise, Qualität kein Thema
- Entwicklung : „Trial-and-Error“
- Produkte eher klein, kurzlebig + undokumentiert
- I.d.R. nur eine Person involviert
- Pflegemaßnahmen + Entwicklung spontan
- Nur für kleine Anwendergruppe gedacht

Professionelle Software-Entwicklung

- „Ad-Hoc-Prozess“ wird nur für experimentelle Entwicklungen + Prototypen eingesetzt
- Problem wenn Produkte aus einer „Ad-Hoc-Entwicklung“ an Benutzer ausgeliefert wird
- I.d.R. geplante + gelenkte Prozesse
  - Abwicklung wird geplant
  - Klare Termin-, Kosten- und Sachziele
  - Ablauf des Prozesses wird überprüft
  - Lenkungsmaßnahmen bei Störungen im Ablauf
  - Qualität ist sehr wichtig
  - Produkt wird gepflegt und i.d.R. weiterentwickelt

Systematische Software-Entwicklung

- Entstehungsprozess mit klar definiertem Anfang und Ende
- Ziel ist es, ein Software-Produkt zu entwickeln, das eine vorgegebenen Aufgabe erfüllt
- Mögliche Optionen
  - Entwicklung neuer Software
  - Beschaffung, Konfiguration und/oder Anpassung

Systematische Software-Pflege (Wartung)

- Kontinuierlicher Prozess
- Ziel: Erhaltung der Gebrauchstauglichkeit
- Regelt Reaktionen auf auftretende Probleme
- Plant + lenkt
  - Weiterentwicklung der Software
    - Zusätzliche Funktionalitäten
    - Änderung bestehender Funktionalitäten (Ablauf)
  - Laufende Anpassung an sich verändernde Umwelt
    - Technische Umgebung (O/S, DB, Netzwerk, ...)
    - Organisatorische Umgebung (funktionale Ergebnisse)

Software-Projekte

- Projekte sind der Abwicklungsrahmen für
  - Systematische (Weiter-)Entwicklung von Software
  - größere, abgrenzbare Pflege-Vorhaben
  - Einführung + Anpassung komplexer SW-Systeme

Projekt (Definition)

ist eine zeitlich befristete Organisationsform zur Bearbeitung einer vorgegebenen Aufgabe und dient zur Regelung der Verantwortlichkeiten und zur Zuteilung der für diese Arbeiten erforderlichen Ressourcen.

Jede systematische Software-Entwicklung und jedes größere Pflegevorhaben werden als Projekt organisiert

Software-Projekte vs. Klassische Projekte

SW Projekte =/= technisches Entwicklungsprojekt

- Wesentlichster Unterschied

  „Software ist immateriell!“

  - Keine natürlichen Ansatzpunkte für Fortschrittskontrolle + Problemerkennung
  - Sorgfältige Planung, Prüfung + Steuerung sehr wichtig
  - Verwendung + Einhaltung geeigneter Prozesse
  - Analyse der Probleme + Risiken
  - Adäquater Umgang mit Projekt-Risiken

Software-Lebenslauf

- Jede isolierte Software-Komponente hat einen „eigenen" Lebenslauf
- Stadien des Lebenslaufs
  - Initiierung - Entwicklung - Nutzung
- Tätigkeiten zur Erstellung von Einzelkomponenten
  - Spezifizieren der Anforderungen + Schnittstellen
  - Konzipieren der Lösung
  - Detail-Entwurf der Lösung
  - Codieren + Testen
  - Integrieren + Testen
  - Installieren + Testen
- Lebenslauf – Zeitraum, in dem eine SW-Komponente bearbeitet oder benutzt wird

Drei Software-Klassen nach Lehmann

- Lehmann teilt Software in 3 Klassen ein
  - *S-Typ*
    vollständig durch formale Spezifikation beschreibbar
    Erfolgskriterium: Spezifikation nachweisbar erfüllt
  - *P-Typ*
    Löst ein abgegrenztes Problem
    Erfolgskriterium: Problem zufriedenstellend gelöst
  - *E-Typ*
    Realisiert eine eingebettete Anwendung
    Erfolgskriterium: Anwender zufrieden
  - Software vom *S-Typ* ist stabil
  - Software vom *P-Typ* und *E-Typ* ist einer Evolution unterworfen

Software-Evolution

„Wandel der Software durch Veränderung des Umfeldes und der Randbedingungen“

- Jedes technische Produkt unterliegt einer Evolution
- Besonderheiten der Informatik
  - Evolution läuft hier besonders schnell ab
  - Teilweise ist die Evolution selbststeuernd

Konsequenzen für Software vom P- / + E-Typ

- *P-Typ* + *E-Typ*-Software ist nie über mehrere Jahre hinweg gebrauchstauglich ohne Veränderung
  - Pflege ist kein „Unfall" sondern unvermeidlich
  - Überlegungen zur Software sollten sich immer auf die gesamte Lebensdauer der Software beziehen
- Längere Software-Projekte:
  - Änderungen der Anforderungen während der Entwicklung sind wahrscheinlich
  - Sich verändernde Ziele sind kein „Unfall" sondern liegen in der Natur der Software-Entwicklung

Meilensteine sind Kontrollpunkte

- Meilenstein
  - eine Stelle in einem Prozess, an dem ein geplantes Ergebnis vorliegt und überprüft werden kann.
  - Das Mittel zur
    - Prozess- und Projekt-Strukturierung
    - Kontrolle des Projektfortschritts
    - Planung durch Festlegung von Ergebnissen
  - Wurde erreicht, wenn das geplante Ergebnis vorliegt
  - Kontrollpunkt zum SOLL-IST-Vergleich
    - Funktionalität
    - Kosten
    - Zeit
  - Nur sinnvoll, wenn messbar / überprüfbar

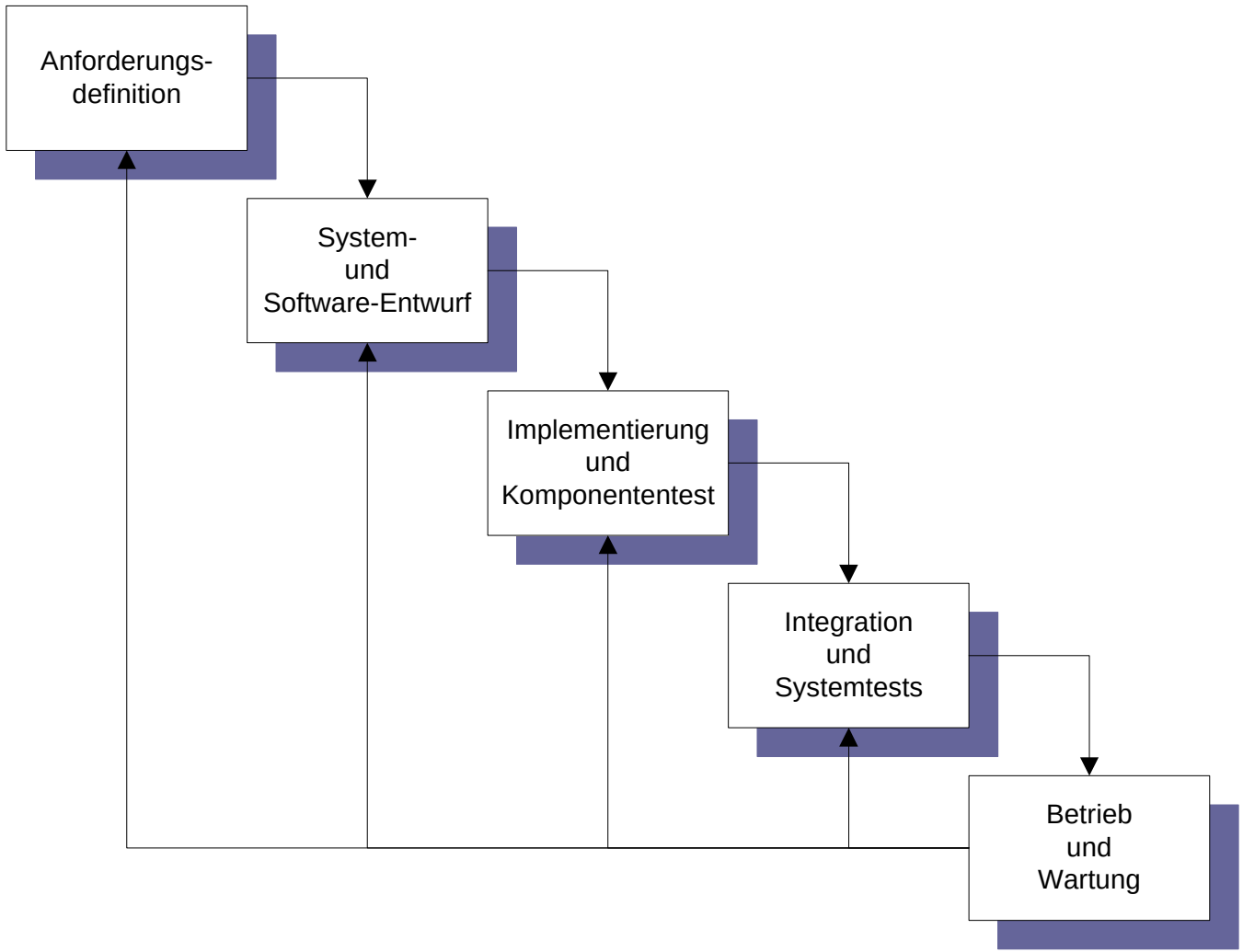
Eignung von Zwischenergebnissen

Ergebnis im SW-Prozess	*Eignung als Meilenstein*
Codierung zu 90 % beendet	UNBRAUCHBAR da nicht messbar, nicht nachweisbar, sondern nur geschätzt
Komponente „abc" hat internen Abnahmetest	GEEIGNET da messbar durch Protokoll des Abnahmetests
Anforderungsspezifikation liegt vor	GEEIGNET wenn Inhalt + Form des Dokuments durch ein „Review" überprüft werden

Was ist ein Software-Prozess?

- Tätigkeiten, durch die Software entsteht
- Vier grundlegende Prozess-Aktivitäten
  - Software-Spezifikation
  - Software-Entwicklung
  - Software-Validierung
  - Software-Evolution
- Eigenschaften des Software-Prozesses
  - systematisch
  - geplant
  - gesteuert

Der Software-Lebenszyklus



Was ist ein Vorgehensmodell?

- Vereinfachte Beschreibung eines SW-Prozesses
- Darstellung aus bestimmter Perspektive
- Abstraktion des tatsächlichen Prozesses
- Arten von Vorgehensmodellen sind
  - Arbeitsablaufmodell
  - Datenfluss- oder Aktivitätsmodell
  - Rollen- / Aktionsmodell
- Modelle in der Software-Entwicklung
  - Wasserfall-Modell
  - Evolutionäre Entwicklung / Wachstumsmodell
  - Formale Systementwicklung
  - Zusammensetzung von Komponenten

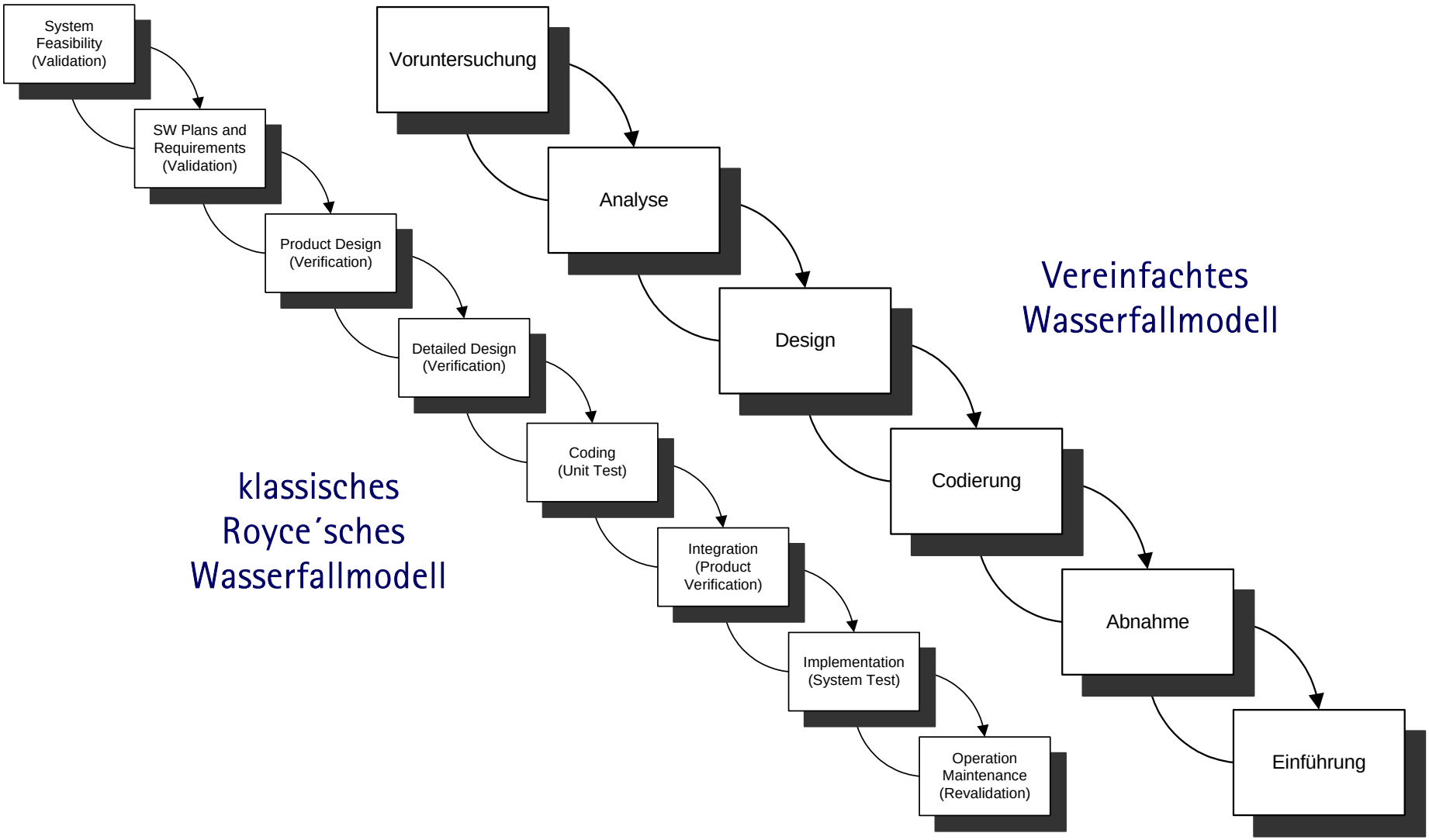
Software-Prozess-Modelle

- Modellvorstellung über den Ablauf der Entstehung eines Software-Produkts / Software-Projekts
  - Welche Schritte sind wie erforderlich?
  - Wie können Zwischenergebnisse überprüft werden?
- Planung und Kontrolle am Modell orientieren
  - Wie gehe ich vor?
  - Was kann ich tun, um das Ziel zu erreichen
- Basis für erfolgreiches Projekt-Management
  - Wie sieht die Planung aus?
  - Was habe ich erreicht – was folgt als nächstes?

Ausgewählte Prozess-Modelle

- Wasserfallmodell
  - Gliederung des Projekts in eine Folge von Aktivitäten
- Ergebnisorientiertes Phasenmodell
  - Gliederung in eine Folge von Zeitabschnitten
    Phasenabschluss durch überprüfbare Meilensteine
- Wachstumsmodell
  - Gliederung in eine Folge von Lieferschritten
- Spiral-Modell
  - Gliederung in eine zyklische, am Risiko orientierte Folge von Entwicklungsschritten
- Prototypen
  - Einsatz in verschiedenen Modellen möglich
    hauptsächlich zur Risikoabschätzung und –Steuerung
- Agile Software-Entwicklung
  - Schlanke Software-Entwicklungs-Prozesse

Wasserfallmodell nach Royce



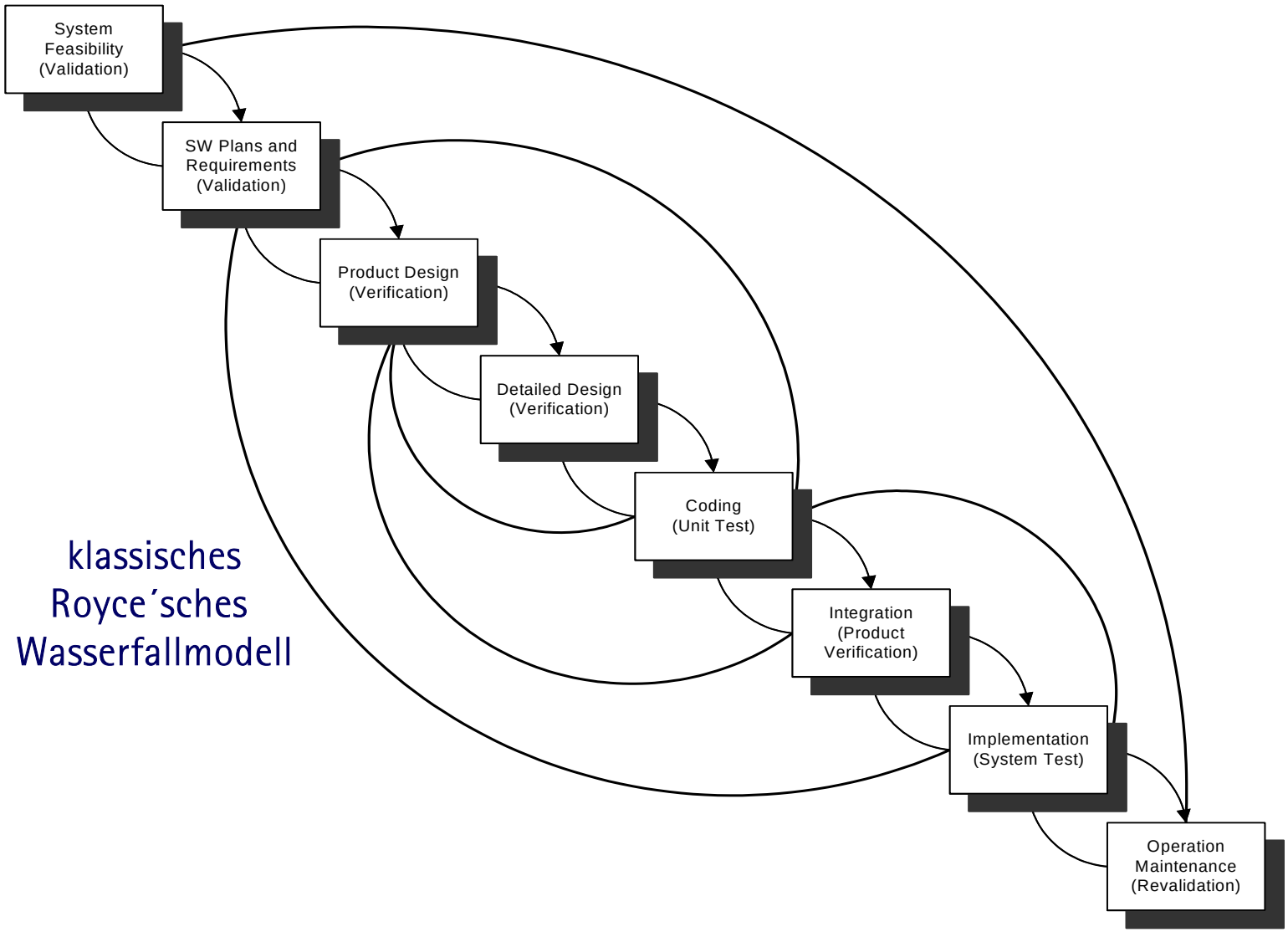
Wasserfallmodell - Eigenschaften

- Ältestes systematisches Prozess-Modell
- Entwicklung als Sequenz aufeinanderfolgender Entwicklungs- und Prüfschritte
- Jede (Haupt-)Tätigkeit bildet eine Phase
- Phasen-Übergang erfolgt, wenn geprüftes und akzeptiertes Ergebnis vorliegt
- Lokale Iterationen sind möglich
- Es gibt eine Vielzahl ähnlich konstruierter Modelle

Hauptproblem:

- Nicht-Lokale Iterationen (Management erschwert)
- In Ursprungsform besser nicht einsetzen!

Wasserfallmodell – „Nicht-lokale Iterationen“

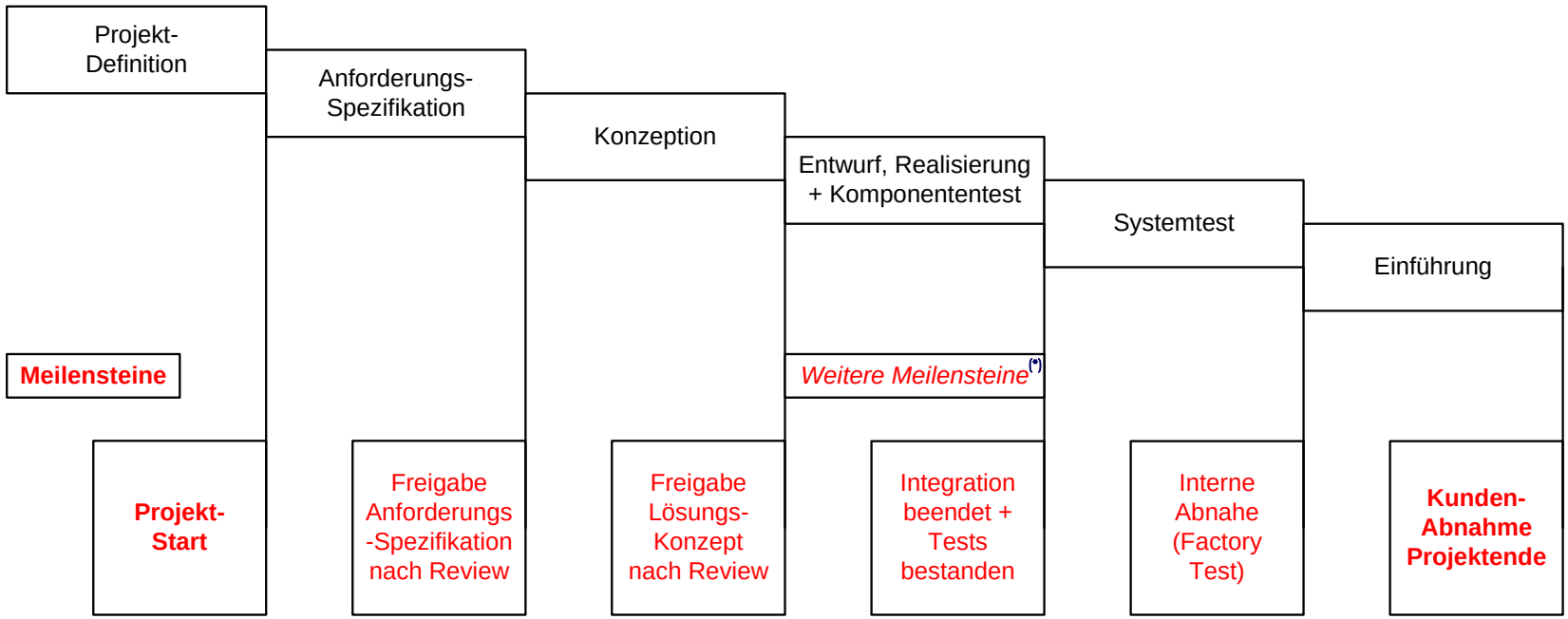


klassisches Royce´sches Wasserfallmodell

Ergebnisorientiertes Phasenmodell –Prinzip

- Orientiert sich am Software-Lebenslauf
- Grundidee:
  Entwicklung wird in eine Sequenz aufeinanderfolgender Zeitabschnitte unterteilt
- Je Phase wird ein Teil des Ergebnisses erarbeitet
- Phase wird nach Ergebnis / Haupttätigkeit benannt
- Keine Iterationen
- Jeder Phasenabschluss ist ein Meilenstein
- Es gibt eine Vielzahl von Varianten und Namen

Ergebnisorientiertes Phasenmodell



(*) Weitere Meilensteine z.B. je Modul nach freigegebenem Entwurf und nach bestandenem Modultest

Ergebnisorientiertes Phasenmodell + / -

- Vorteile
  - \+ Leicht verständlich
  - \+ Folgt dem „natürlichen" Lebenslauf
  - \+ Geeignet für Projekt-Management
  - \+ Fördert planvolles Vorgehen
  - \+ Trägt zur frühzeitigen Fehlererkennung bei
- Nachteile
  - – Ignoriert die Software-Evolution
  - – Lauffähige Systeme entstehen erst sehr spät
    - – Lange „Papier-Durststrecke"
    - – Technische Risiken
    - – Adäquatheitsrisiken
  - – System wird auf einen Schlag in Betrieb genommen (Big Bang)

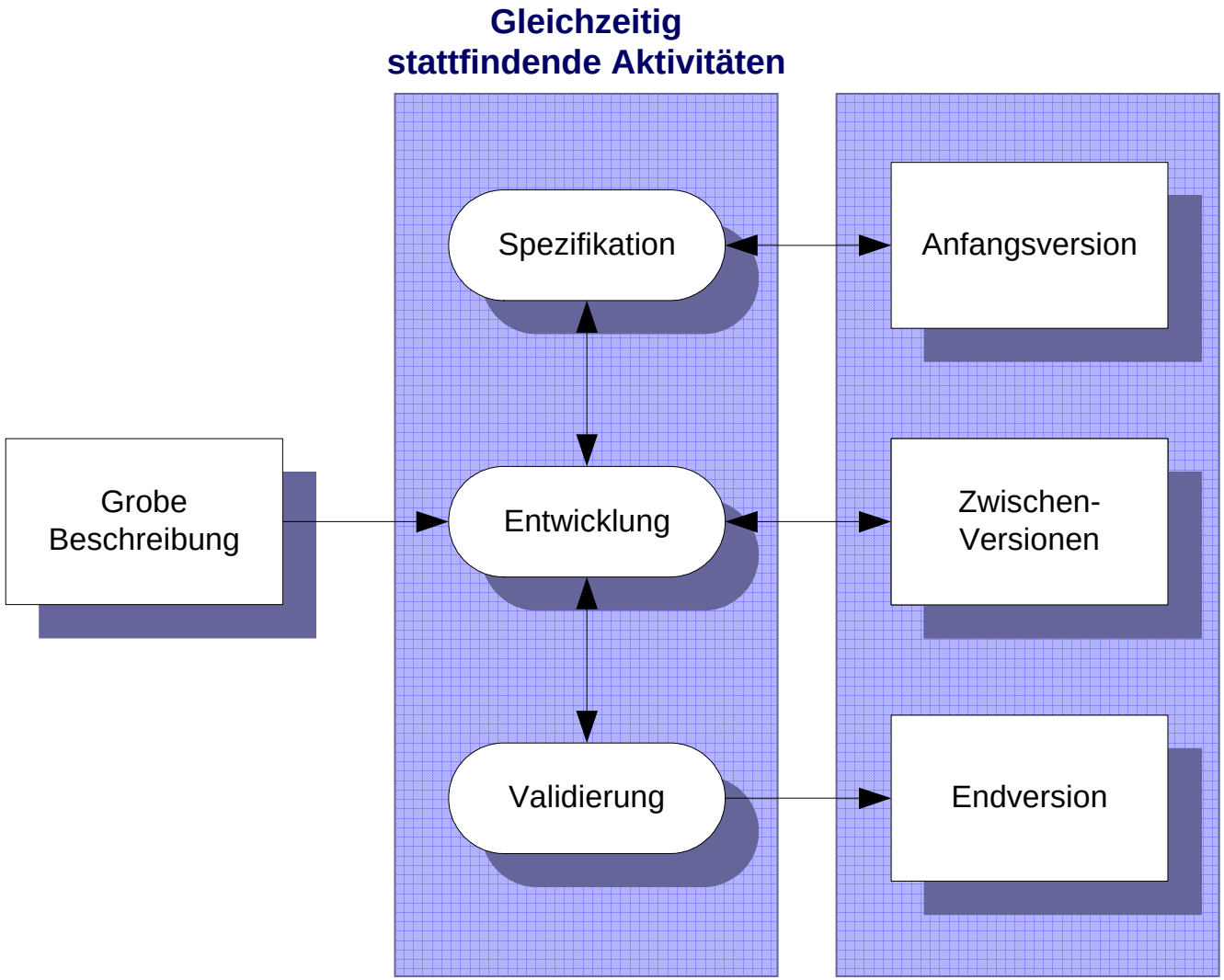
Ergebnisorientiertes Phasenmodell - Eignung

- Ergebnisorientierte Phasenmodelle sind geeignet
  - Management nicht zu großer Projekte
  - Projekte mit genau definierten Anforderungen
  - Wenn die Entwickler über das erforderliche „Know-How“ verfügen
  - Wenn das Entwicklungsrisiko eher gering ist

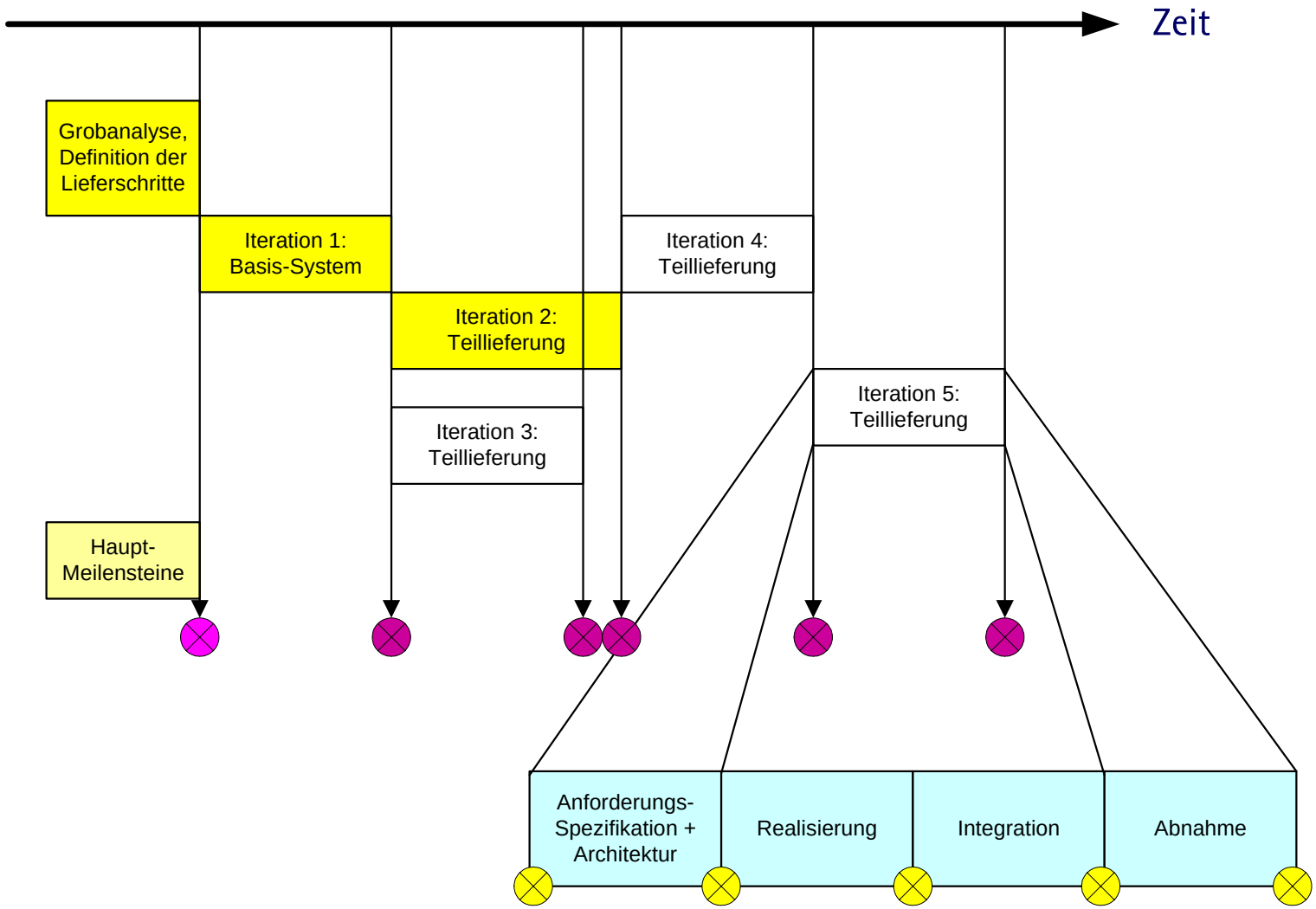
Evolutionäre Entwicklung - Wachstumsmodell

- Leitbild: Software-Evolution
- System wird nicht konstruiert, sondern wächst
- Verschiedene Bezeichnungen
  - Versionen-Modell
  - Evolutionäres Modell
  - Inkrementelles Modell
- Grundidee:
  Gliederung der zu liefernden Software in aufeinander aufbauende, betriebsfähige Lieferungen
- Unterteilt Entwicklung in Folgen von Iterationen
- Pro Iteration: Vollständiges Teilergebnis mit betriebsfähiger Software
- Lieferung sind als autonome Teilprojekte organisiert

Evolutionäre Entwicklung - Wachstumsmodell



Evolutionäre Entwicklung - Beispiel



Evolutionäre Entwicklung - Planung

- Umfang und Reihenfolge der Lieferungen
- Planung bis zum gewünschten Endzustand
- Schrittweiser Ausbau nach verschiedenen Merkmalen ist möglich
  - Nach Funktionalität
  - Nach Bedienungskomfort
  - Nach Leistung + Performance
  - Nach Verwendbarkeit

Evolutionäre Entwicklung – Integration

- Normales Wachstumsmodell
  - Betriebsfähige Version des Systems nach jeder Teillieferung
- Kontinuierliche Integration
  - Ständig eine betriebsfähige Referenz-Version mit dem aktuellen Entwicklungsstand verfügbar
- Referenzversion wird in kurzen Abständen (täglich, wöchentlich, ...) neu erzeugt („daily build")
- Verkürzt die Rückkopplungszyklen und senkt damit die Fehlerkosten
- Funktioniert gut, wenn
  - Aufgabe in viele kleine, autonome Teilaufgaben teilbar
  - Aufwand für Neuerzeugung der Referenzversion gering ist
- Gefahr, dass notwendige Restrukturierungen unterbleiben

Evolutionäre Entwicklung + / -

- Vorteile
  - \+ Modellieren das natürliche Verhalten von Systemen
  - \+ Sehr gut geeignet für Projekt-Management
  - \+ Frühe Verfügbarkeit lauffähiger Teilsysteme
    - \+ Fördert die Motivation der Beteiligten
    - \+ Lindert größte Nöte durch System-Verfügbarkeit
  - \+ Verkürzte Rückkopplungszyklen
  - \+ Schrittweise System-Einführung
- Nachteile
  - – Gefahr der Insel-Bildung
  - – Gefahr der Zerstörung von Strukturen und Konzepten durch den schrittweisen Ausbau des Systems

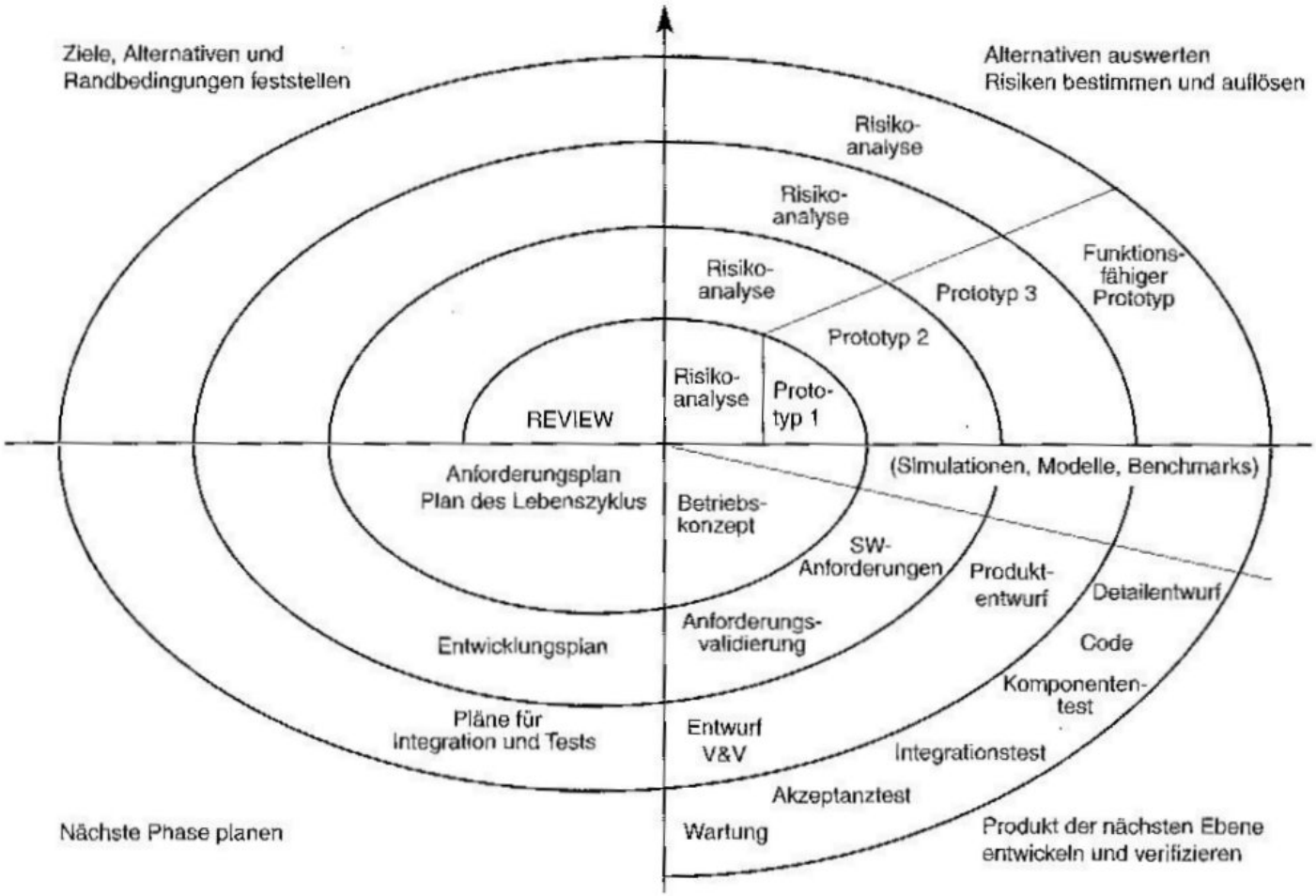
Evolutionäre Entwicklung - Eignung

- Evolutionäre Entwicklung ist gut geeignet bei
  - großen Systemen mit langen Entwicklungszeiten
  - Systemen, bei denen die volle Funktionalität nicht sofort verfügbar sein muß
  - Vorhaben, die vorab nicht vollständig spezifiziert sein müssen
  - Ein erstes Ergebnis und Basis-System schnell verfügbar sein soll

Spiralmodell von Boehm (1988)

- Weiterentwicklung des Wasserfallmodells
- Vor allem für die Entwicklung großer und risikoreicher Projekte / Systeme gedacht
- Zweigeteilter Zyklus (Entwickeln / Prüfen) des Wasserfallmodells wird durch vierteiligen Zyklus ersetzt
  - Planung
  - Zielsetzung / Steuerung / Zielkorrektur
  - Risikountersuchung / Risikobewertung
  - Entwicklung + Prüfung
- Manchmal wird das Spiralmodell fälschlicherweise als Wachstumsmodell bezeichnet

Spiralmodell von Boehm



Agile Software-Entwicklung - Idee

- Idee
  Die Verfahren für die Entwicklung von Klein-Software so auf große Software-Projekte übertragen, dass
  - Es erfolgreich funktioniert
  - Die Prozesse möglichst einfach und wenig reglementiert sind (Zeitersparnis, weniger Prozess-Bürokratie)
  - Entwickelt in den späten „90er" Jahren, ab 2000 zunehmend populär

Agile Software-Entwicklung - Prinzipien

- Grundlegende Prinzipien
  - Je kleiner die Teilaufgabe und je kürzer die Rückkopplungszyklen, desto besser
  - Der Kunde gestaltet das Projekt während seiner Entstehung
    - Kundenvertreter ist/sind aktiv am Projekt beteiligt
  - Je freier und konzentrierter die Entwickler arbeiten können, desto besser
  - Die Qualität wird an der Quelle sichergestellt
    - „pair programming“
    - Rigorose Komponententests
  - Keine Arbeit auf Vorrat

Agile Software-Entwicklung - Erfahrungen

- Erste Erfahrungen
  Agile Software-Entwicklung funktioniert, wenn
  - Auftraggebervertreter verfügbar, kompetent und entscheidungsbefugt sind
  - Das Problem in hinreichend viele kleine Teilaufgaben unterteilbar ist
  - Ein kleines Entwicklungs-Team
    (oder mehrere kleine parallele Teams)
  - Ein kompetenter System-/Software-Architekt
  - Kontinuierliche Integration mit geringem Aufwand möglich ist
  - Konsequente Qualitätssicherung an der Quelle erfolgt
- Sonst Absturz ins Chaos und in Ad-Hoc-Entwicklung
- Vor- und Nachteile wie bei Wachstumsmodell, aber verstärkt

Prototyp und Prototyping

- Zentrales Mittel zur frühzeitigen Erkennung und Lösung von Problemen
- Prototyp
  Lauffähiges Stück Software, welches kritische Teile eines zu entwickelnden Systems vorab realisiert
- Prototyping
  Prozess zur Erstellung und Nutzung von Prototypen
- Drei Arten von Prototyping
  - Explorativ
  - Experimentell
  - Evolutionär

Exploratives + experimentelles Prototyping

- Explorativ
  - Klärung und Bestimmung von Anforderungen
  - Demonstrations-Prototypen
    Demonstration der prinzipiellen Machbarkeit und Nützlichkeit von Systemideen
  - Prototyp im engeren Sinn
    Erprobbares und kritisierbares Modell einer geplanten Informatik-Lösung zur Überprüfung der
    - Angemessenheit von Anforderungen
    - Tauglichkeit vorgesehener Lösungen
- Experimentell
  - Entwicklung von Labormustern zur
    - Untersuchung der Realisierbarkeit kritischer Systemteile
    - Bewertung von Entwurfsalternativen

Exploratives + experimentelles Prototyping

- Demonstrations-Prototypen, Prototypen im engeren Sinn und Labormuster sind „Wegwerf-Prototypen“
  - Dürfen undokumentiert sein
  - Dürfen schnelle, Software-technische unsaubere Lösungen verwenden
  - Sind wirtschaftlich, wenn
    - Aufgrund der gewonnen Erfahrung mehr Entwicklungskosten eingespart werden, als der Prototyp gekostet hat
    - Ein gefährliches Risiko wesentlich gemindert wird
  - Das tatsächliche Wegwerfen des Prototyps sichergestellt ist (sonst Ad-Hoc-Entwicklung!)

Evolutionäres Prototyping

- Prototyp ist Pilotsystem
- Bildet den Kern des zu entwickelnden Systems
- Kein Wegwerf-Prototyp
- Muss nach den Regeln der Kunst entwickelt werden
  - Planung + Steuerung
  - Überprüfung / Reviews
  - Dokumentation
- Evolutionäres Prototyping entspricht einem Wachstumsmodell

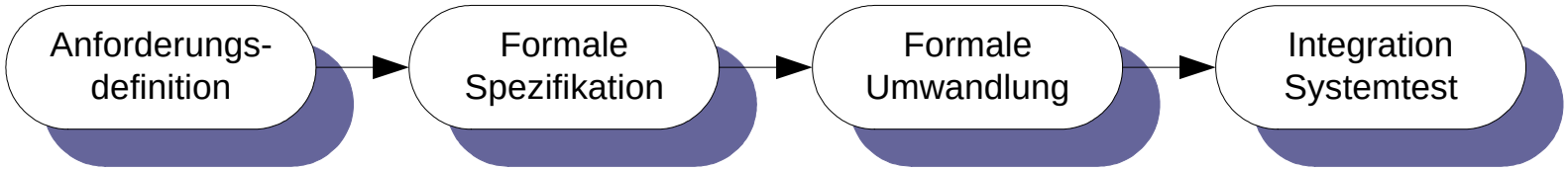
Prototyping und Prozessmodelle

- Prototyping ist kein eigenständiges Prozessmodell
  - Demonstrations-Prototypen
    - Unterstützen Akquisition und Aufsetzen von Projekten
    - Geben einen ersten Eindruck des Systems
  - Prototypen im engeren Sinn + Labormuster
    - Dienen der Risiko-Minderung
    - Sind in jedem Prozessmodell jederzeit einsetzbar
  - Entwicklung von Pilotsystemen
    - Eine Form des Wachstumsmodells

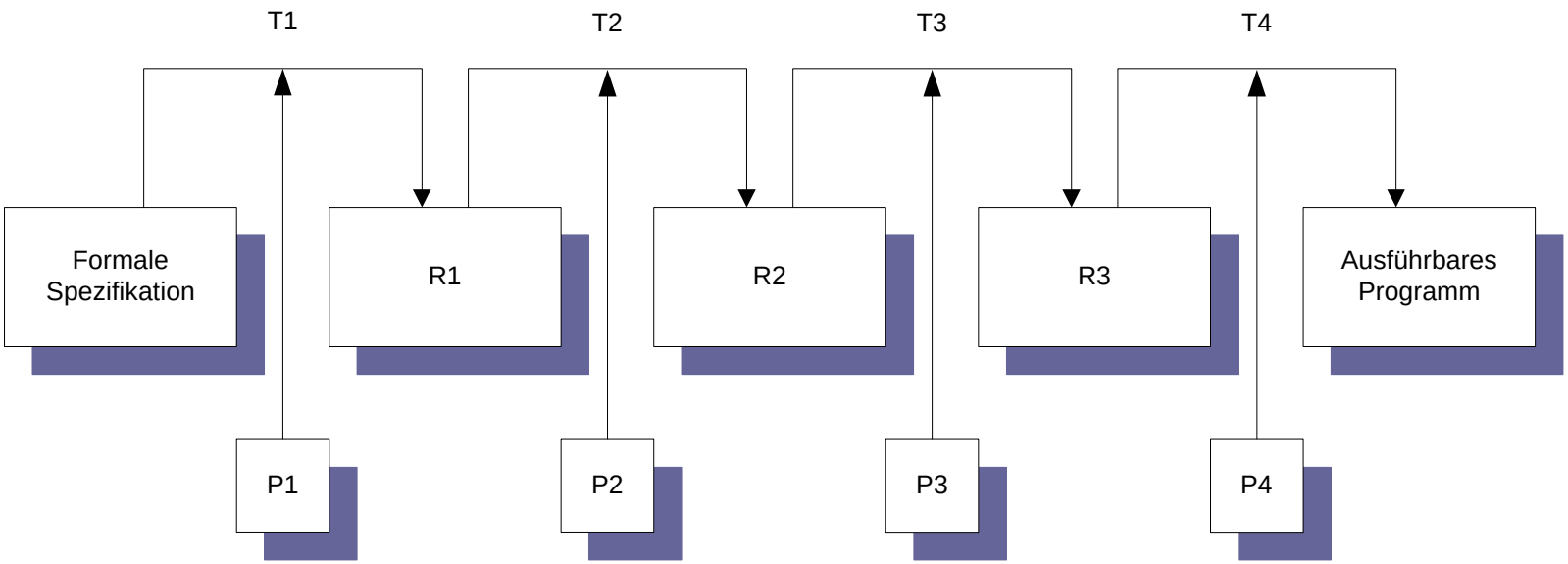
Weitere Prozessmodelle

- Formale Systementwicklung
- Wiederverwendungsorientiertes SE
- Inkrementelle Entwicklung

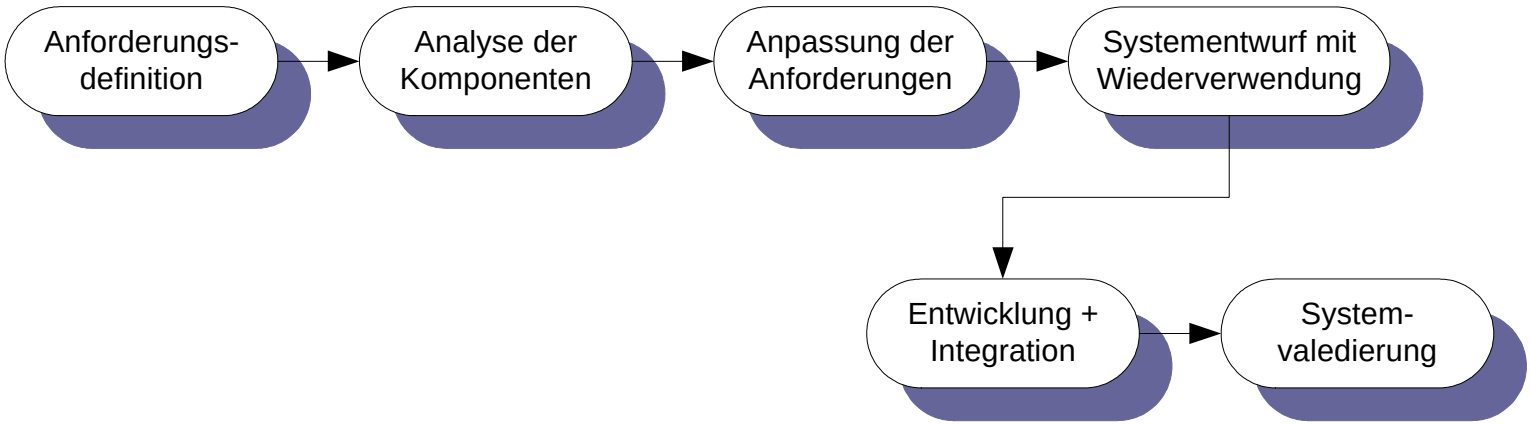
Formale Systementwicklung



Formale Umwandlungen

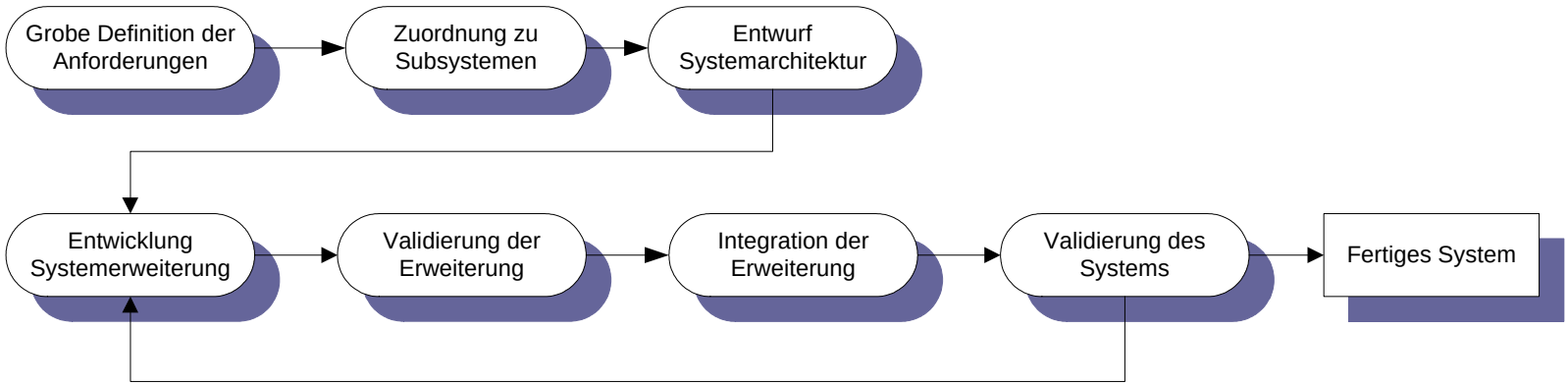


Wiederverwendungsorientiertes SE



- Analyse der erforderlichen Komponenten
- Prüfung bestehender Bibliotheken auf Verwendbarkeit
- Systementwurf als „Schichten-Architektur" (s. UNIX)
- Zusatzentwicklungen + Integration der Komponenten
- Test und Verifikation des Systems

Inkrementelle Entwicklung



- Basis-Implementierung des Grundsystems
- Zunehmende Verfeinerung der Basis-Funktionen
- Zunehmende Erweiterung der Basis-Funktionen
- Überprüfung der Korrektheit nach jedem Schritt
- System-Iteration bis hinreichende Funktionalität + Qualität

Wiederverwendung + Beschaffung

- Ziel

  Kosten senken für Software durch
  - Entwicklungs-Produktivität steigern
  - Möglichkeiten begrenzen
  - Stückzahl erhöhen
    - Das Geheimnis der billigen Hardware
    - Bei Software durch
      - Wiederverwendung
      - Beschaffung

Wiederverwendung

- Wiederverwendbare Software muss erst einmal geschaffen werden
  - Komponenten bilden, entwickeln + dokumentieren
- Entwicklung wiederverwendbarer Software ist teurer als die zweckgebundene Software
  - Explizite Anreize für Entwicklung wiederverwendbarer Software
- Wiederauffinden von Wiederverwendbarem
  - Kataloge + Beschreibungen
- Pflege von wiederverwendbarer Software
  - Pflege- / Wartungsverträge
- Förderung der Wiederverwendung
  - Handel mit Komponenten
  - Informationsstelle für Wiederverwendung

Beschaffung – „Make or Buy"

- Wo immer möglich, denn
  fast immer günstiger als Eigenentwicklung
  (Stückzahl)
- Unbedingt Berücksichtigung der
  - Anpassungs- und Parametrisierungs-Kosten
  - Wirtschaftlichkeit im Einzelfall prüfen
- Beschaffungsprozess ist eingebettet in Entwicklungsprozess
  - Spezifikation, Test, Installation ist auch bei Beschaffung unbedingt erforderlich
  - Hauptaktivität für Beschaffung in der Phase der Konzeption der Lösung

„Make or Buy“ im Entwicklungsprojekt

Die Beteiligten müssen mitmachen

- Nicht alles selbst machen wollen
- Nicht das Perfekte (und Teure), sondern das Gebrauchstaugliche schaffen
- Mitarbeiterinnen und Mitarbeiter nicht nach der produzierten Software-Menge beurteilen
- Suche nach „Beschaffbarem“ und „Wiederverwendbarem“ zum selbstverständlichen Bestandteil jedes Entwicklungsprojekts machen

Software-Pflege - Wartung

- Pflege (Wartung, „Maintenance“)
  Modifikation und / oder Ergänzung bzw. Erweiterung bestehender Software durch veränderte Software mit dem Ziel, Fehler zu beheben und das Produkt an veränderte Bedürfnisse oder Umweltbedingungen anzupassen oder die bestehende Software um neue Fähigkeiten zu erweitern
  - Systematische Erhaltung der Gebrauchstauglichkeit von Programmen
  - Nicht nur reine Fehlerbehebung
  - Bei Software vom P-Typ und E-Typ (nach Lehmann) zwingend
  - Kontinuierlicher Prozess über die gesamte Lebenszeit der Software

Der Pflege-Prozess

Kontinuierlicher Prozess mit folgenden Teilaufgaben:

- Erfassung der Kundenbedürfnisse
  - Reaktiv: Problem-/Fehlermeldung
  - Proaktiv: Weiterentwicklung / Anpassung an veränderte Bedingungen
  - Analyse der Kundenbedürfnisse
  - Planung der Pflegearbeiten, Festlegung von Arbeitspaketen
    - ↳ Release / Freigabe
  - Geplante Auslieferung der Ergebnisse
  - Verwaltung der Meldungen und Änderungen
    - ↳ Konfigurations- / Versionsmanagement

Release / Version / Patch

- Sofortige Inbetriebnahme / Auslieferung jeder Änderung ist unmöglich / gefährlich
- Zusammenfassung von Änderungen in „Arbeitspakete" erforderlich
  - „Release" / „Version" oder „Patch"
- Release / Version
  Eine konsistente Menge von Komponenten eines Systems, die gemeinsam zur Benutzung freigegeben werden
- Patch
  Fehlerbehebung durch gezielte Nacharbeit und Austausch fehlerhafter Module eines Systems
- Komponenten
  Programm-Module, Daten, Steuerdateien, Dokumente
  - Identifikation erfolgt durch Nummerierung (z.B. 2.4, 3.7.5)
    - 1. Ziffer Haupt-Version / Major-Release (alle 1-3 Jahre)
    - Folgeziffern Unterversion / Unter-Release (alle paar Monate)
      (gerade: Funktions-Release, ungerade: Fehler-Release)

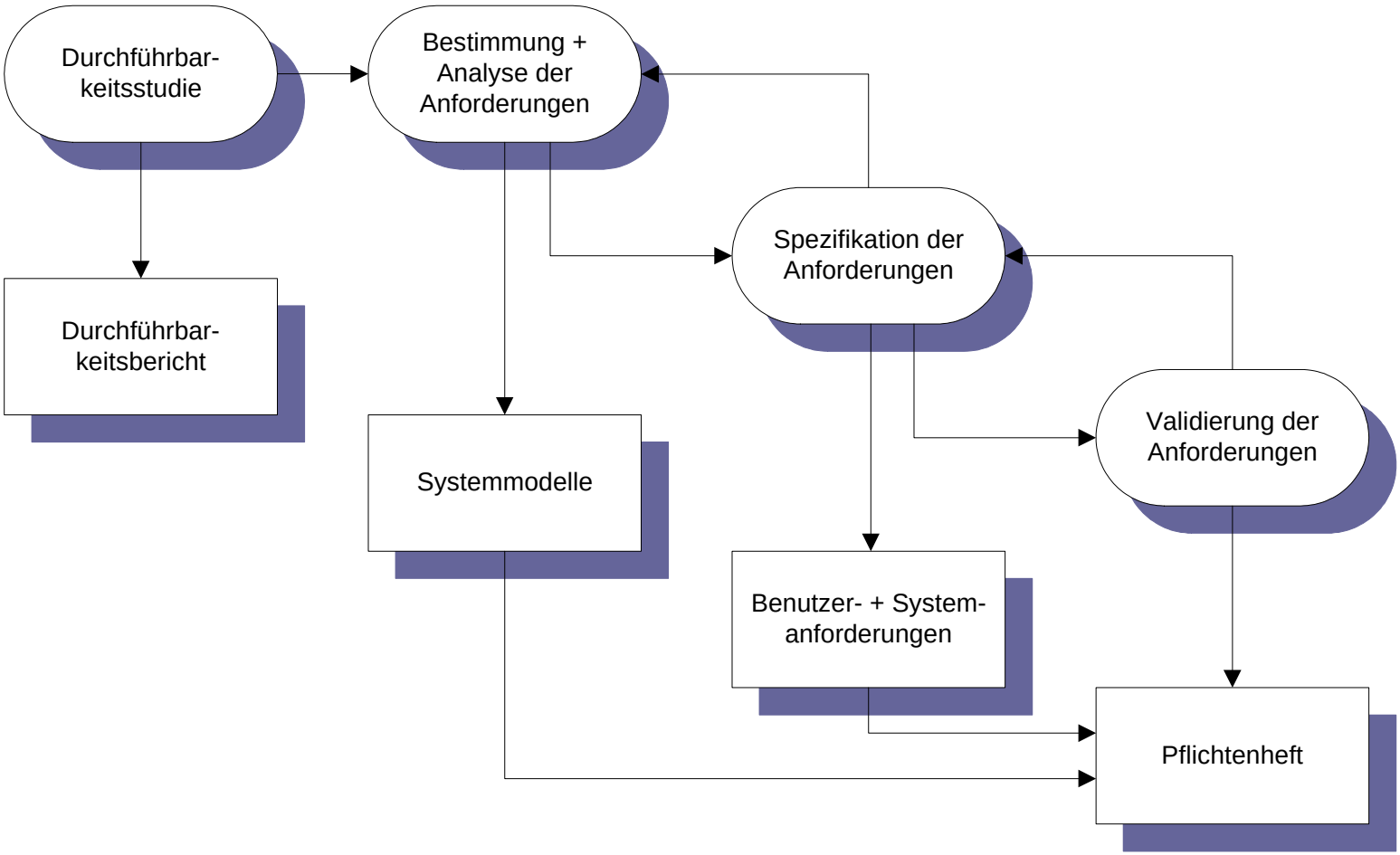
Veränderungen durch Software-Pflege

- Pflege ist eine notwendige Folge der Software-Evolution
- Durch Pflege wird Software größer, umfangreicher aber dadurch auch schlechter und intransparenter
- Die Software-Evolution unterliegt einer Eigendynamik
  - Software-Pflege ist nur bedingt steuerbar
- Software-Gesetze von Lehmann (Typ P + E)
  1. Gesetz der kontinuierlichen Veränderung
  2. Gesetz der zunehmenden Komplexität
  3. Grundgesetz der Programm-Evolution
  4. Gesetz der Invarianz des Arbeitsaufwands
  5. Gesetz der Invarianz des Release-Inhalts
  6. Gesetz des kontinuierlichen Wachstums

Anforderungsanalyse

- Ziele der geplanten Software-Entwicklung klären
  - Management / Auftraggeber
  - Anwender / Betreuer
- Funktionalitäten mit den Anwendern erarbeiten
  - Fragenkatalog
  - Referenz-Systeme
- Abläufe, Datenmodell und Integration
  - Aufnahme der Arbeitsschritte Optimierung
  - Datenmodell erarbeiten
    - Erforderliche Daten und Zusammenhänge
    - Datenmengen und Verfügbarkeit / Datenübernahme
  - Integrationsanforderungen in andere Anwendungen

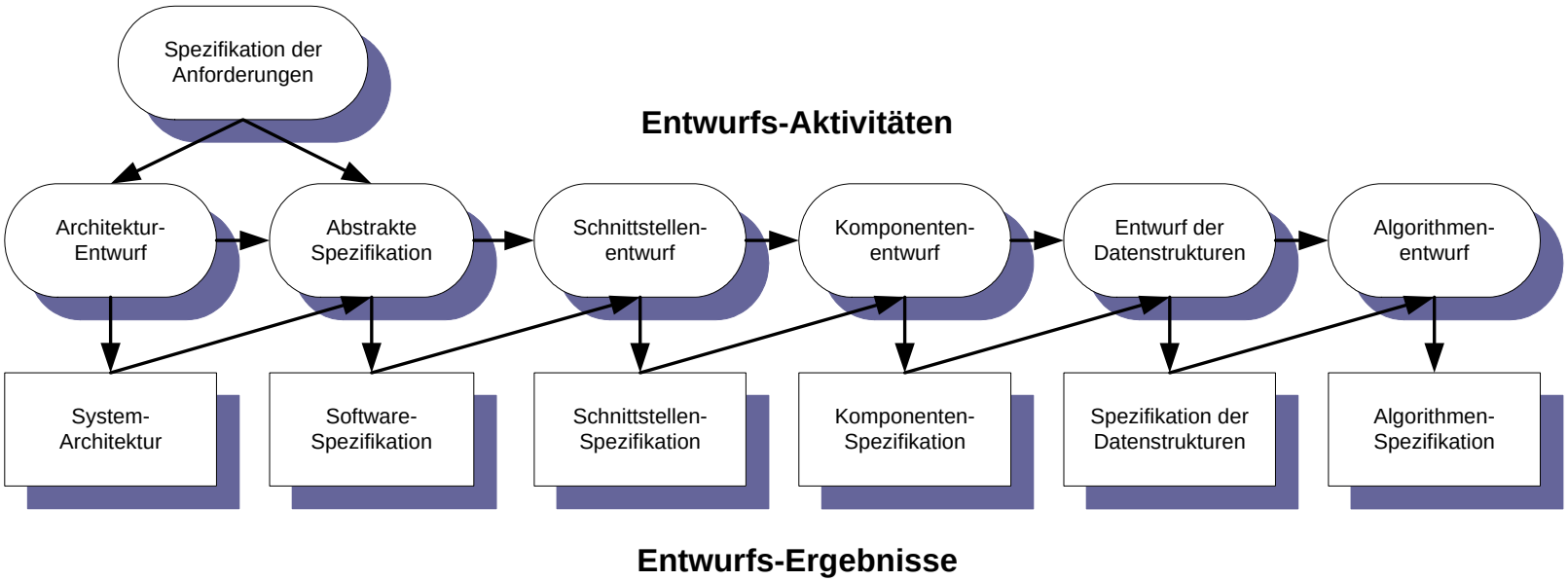
Ablauf der Anforderungsanalyse



Phasen der Anforderungsanalyse

- Durchführbarkeitsstudie
  - Abschätzung, ob Bedürfnisse mit aktueller SW / HW in vertretbarem Rahmen erfüllbar sind
  - Kurzfristige + schnelle Entscheidungsgrundlage
- Bestimmung + Analyse der Anforderungen
  - Beobachtung bestehender Referenzsysteme
  - Diskussion + Befragung der Anwender (evtl. Prototyp)
- Spezifikation der Anforderungen
  - Umsetzung der Anforderungen in klare Beschreibungen
  - Erstellung des Pflichtenhefts + Detailbeschreibung
- Validierung der Anforderungen
  - Prüfung ob Anforderungen realistisch, konsistent + vollständig
  - Ergänzung der Anforderung bis zur Abnahme

Allgemeines Modell des Entwurfsprozesses



- Entwurf erstellt die Basisinformation für Implementierung
- Beschreibung der Struktur
  - Des Ziel-Systems / Software-Produkts
  - Daten und Datenabhängigkeiten
  - Schnittstellen und Übergangs-Mechanismen
  - Algorithmen + Verarbeitungsschritte

Entwurfsprozess

- Mehrere Modelle auf unterschiedlichen Abstraktionsebenen
  - Fehler + Lücken im bestehenden Entwurf beseitigen
  - Frühzeitige Überprüfung + Verbesserung des Modells
  - Phasen miteinander verknüpft und beeinflussen sich gegenseitig
- Ergebnis einer Phase ist Spezifikation für Folgephase
- Gesamtergebnis ist eine detaillierte Spezifikation der Funktionen, Datenstrukturen + Algorithmen
- Aktivitäten sind dabei (in Realität vermischt!)
  - Architektur-Entwurf mit Sub-Systemen + Abhängigkeiten
  - Abstrakte Spezifikation mit Funktionen + Randbedingungen
  - Schnittstellen-Entwurf – eindeutig + konkret
  - Komponenten-Entwurf mit Schnittstellen
  - Entwurf der Datenstrukturen und deren Abhängigkeiten
  - Algorithmen-Entwurf im Detail für Funktionen

Entwurfsmethoden

- Oft „Ad-Hoc-Prozess“
  - Willkürliche Sammlung von Anforderungen (häufig in natürlicher Sprache)
  - Informeller Entwurf des Systems
  - Basis für die Programmierung (oft ungenügend!)
  - Veränderung der Anforderungs-Spezifikation während der Umsetzung
  - Nach Systemabnahme ist Ursprungs-Spezifikation meist ungültig durch häufige Änderungen
    - Inkorrekt
    - unvollständig

Entwurfsmethoden

- Strukturierte Methoden für den Systementwurf
  - Strukturierter Entwurf nach Constantin + Yourdon 1979
  - Strukturierte Systemanalyse nach Gane + Sarson 1979
  - Jackson System Development nach Jackson 1983
  - Methoden der objektorientierten Entwicklung
    z.B. nach Robinson 1992; Booch 1994; Rumbaugh 1991 u.a.
- Erstellen grafischer Systemmodelle + Entwurfsdokumente
- Einsatz von CASE-Werkzeugen für bestimmte Methoden
- Strukturierte Methoden werden erfolgreich eingesetzt
- Reduktion der Kosten weil Standards verwendet werden
- Kaum Unterschiede bei strukturierten Methoden
- Erfolg hängt von Eignung bzgl. des Einsatzes ab

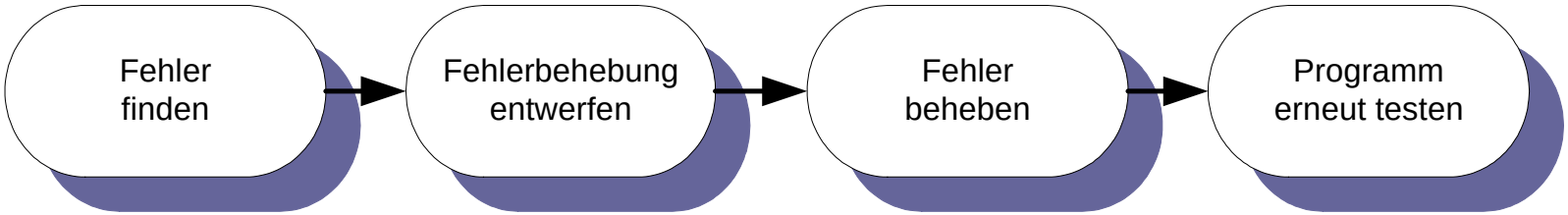
Strukturierte Methoden

- Umfang der strukturierten Methoden
  - Modell des Entwurfsprozesses
  - Notation zur Darstellung des Entwurfs
  - Berichtsformate
  - Regeln + Entwurfsrichtlinien
- Unterstützung für
  - Datenfluss-Modell mit Datenumwandlung
  - „Entity-Relationship-Modell“
  - Strukturelles Modell der Komponenten + Interaktionen
  - Objektorientierte Methoden
    - Vererbungsmodell
    - Modell der statischen + dynamischen Beziehungen
    - Modell bezgl. des Zusammenspiels der Objekte

Strukturierte Methoden

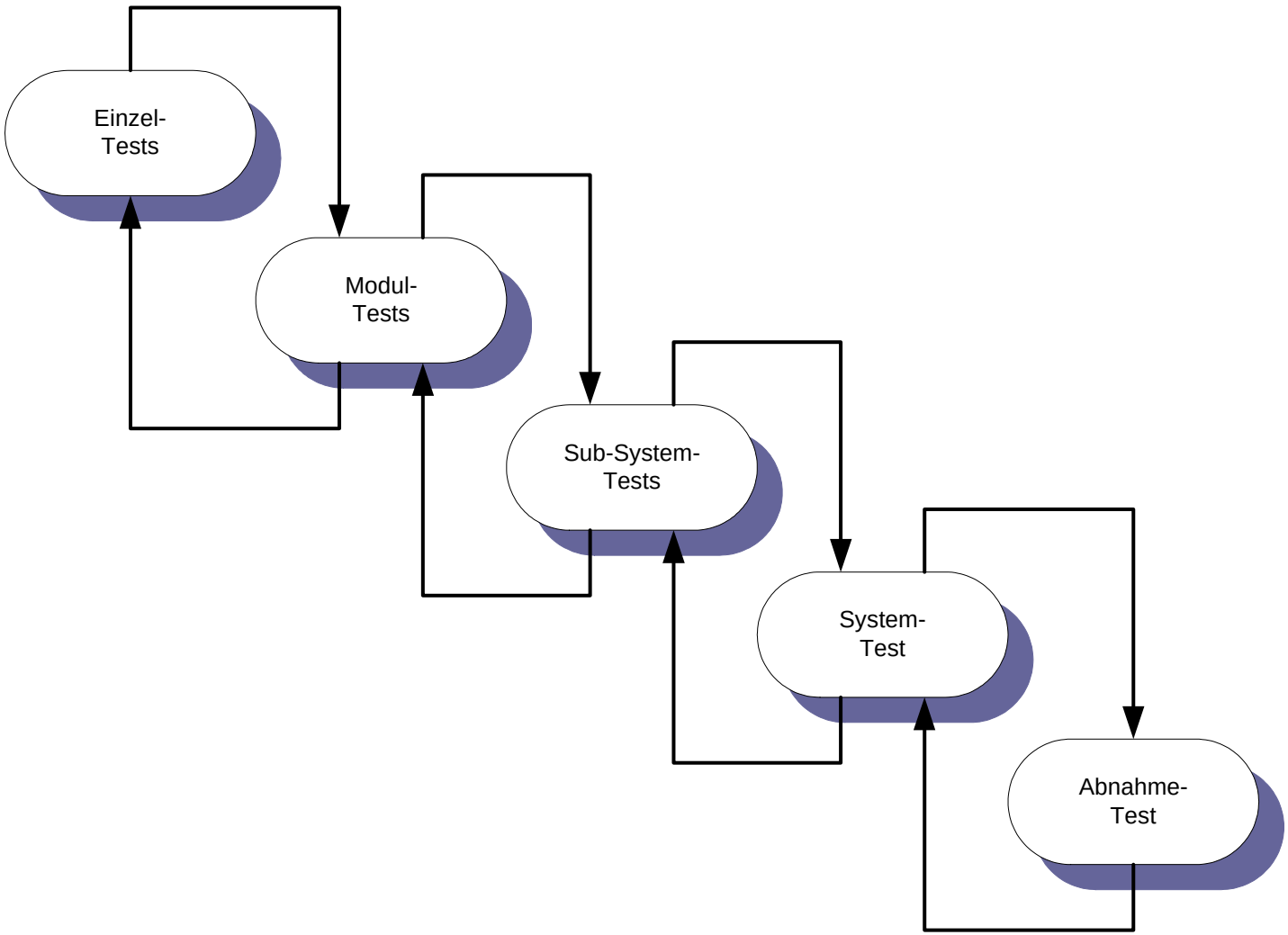
- Zusätzliche Modelle
  - Zustandsdiagramme
  - Zentrales „Data Dictionary“
- Praxis
  - Anleitung eher informell
  - Daher unterschiedliche Entwürfe verschiedener Entwickler
  - Methoden sind in Wirklichkeit „Standard-Notationen“ guter Entwicklungspraxis
  - Bei Strukturierung ist Kreativität sehr wichtig
  - Eigentlich nur Anleitung + Richtlinien mit Adaption

Programmierung + Fehlerbehebung



- CASE-Werkzeuge für einen Programmierrahmen
- Programmierung der Details der Komponenten
- Programmierung - individueller + kreativer Prozess
- Keine definierte Vorgehensweise für Programmierung
- Basistests der entwickelten Module durch Entwickler
- Tests möglichst vollständig / alle Programm-Zweige
- Hypothesen bzgl. des Programmverhaltens
- Aufbau einer Testumgebung/ Testrahmen
- Einbau von Testpunkten in den Modulen (abschaltbar)
- Endtest der Module + Komponenten durch Unbeteiligte

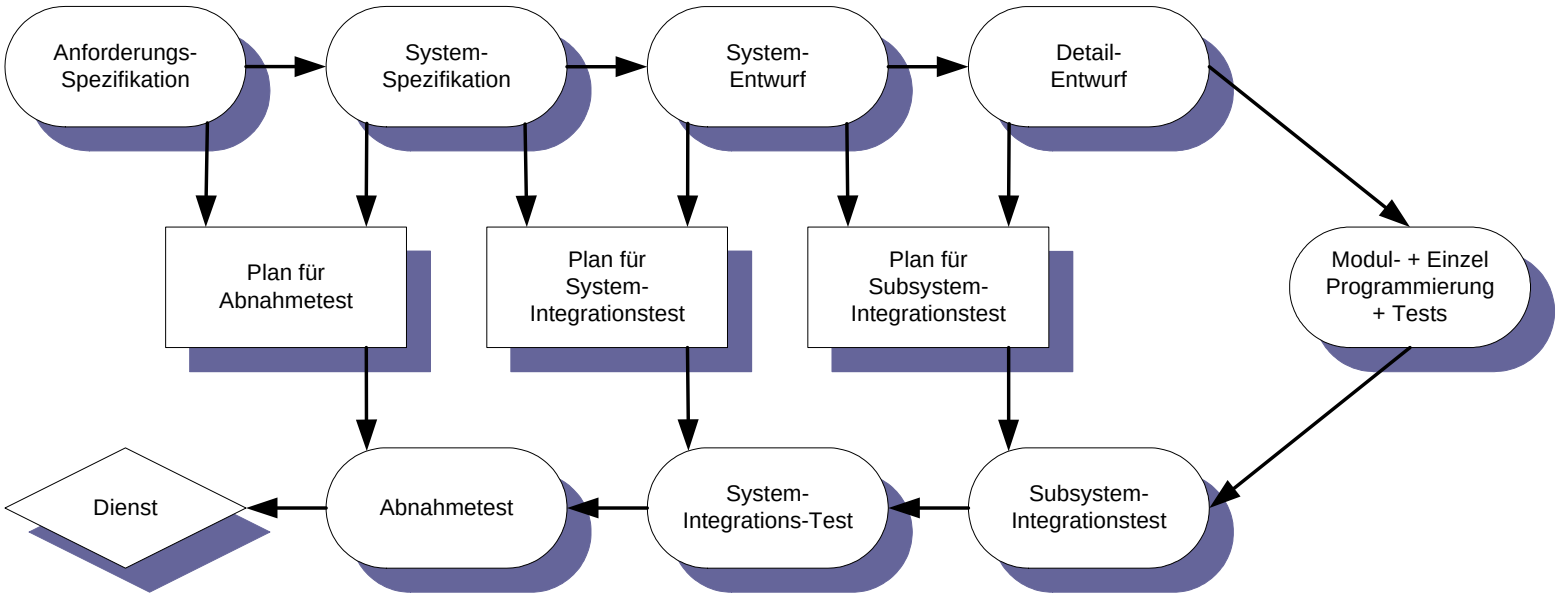
Software-Validierung



Software-Validierung

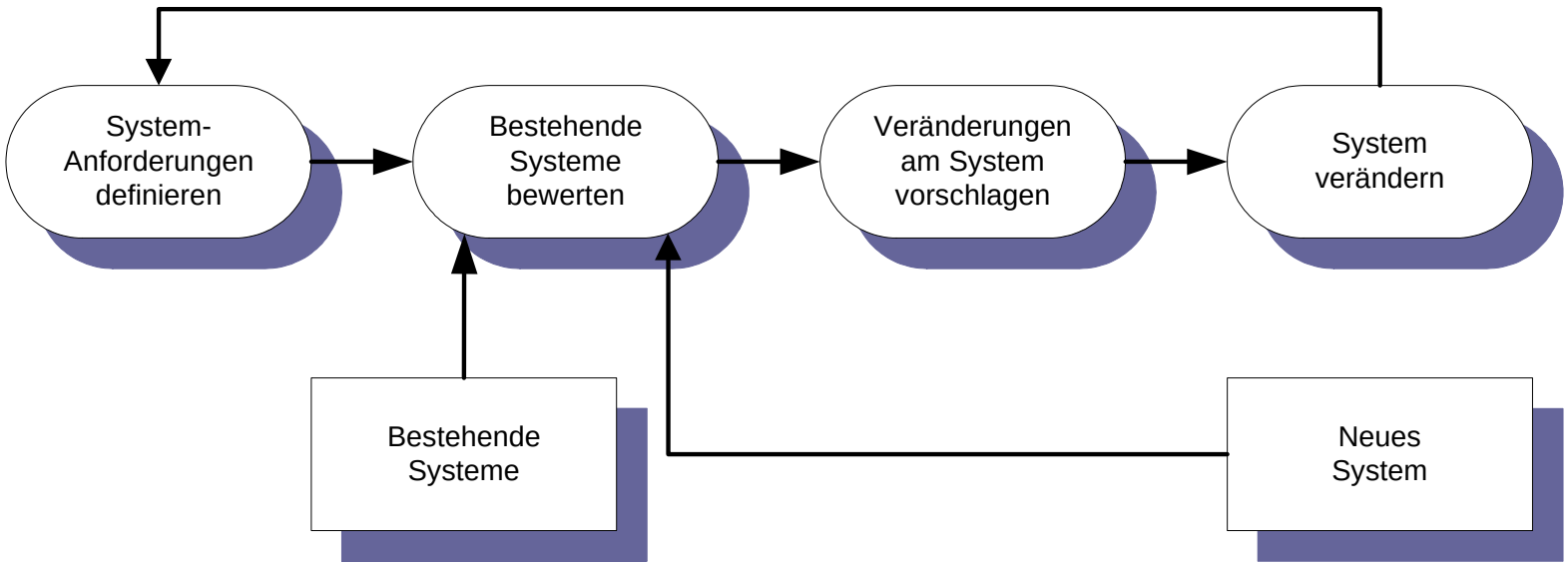
- Verifikation + Validation (V+V)
  - Zeigt den Erfüllungsgrad der Spezifikation
  - Zeigt die Erfüllung der Kunden-Erwartungen
  - Sieht Überprüfungsprozesse in jeder Phase vor
  - Hauptanteil beim Systemtest / Abnahmetest
- Phasen des Testprozesses
  - Einzeltests der Komponenten
  - Modultest – Test zusammenhängiger Komponenten
  - Subsystem-Tests –zusammenhängender Module
  - Systemtest – Test des Gesamtsystems
  - Abnahmetest – letzte Testphase mit dem Anwender

Testphasen im Software-Prozess



- Einzel- und Modultest liegen beim Programmierer
- Spätere Testphasen betreffen die Integration der Komponenten
- Test der weiteren Phasen durch unabhängiges Team von Testern
- Testpläne vorher spezifizieren und Daten beschreiben
- Abnahmetest wird auch Alpha-Test genannt (↳ Alpha-Version)

Weiterentwicklung von Software



- Trennung zwischen Neuentwicklung + Wartung verschwindet
- Kaum Neuentwicklungen – meist Weiterentwicklungen
- Weiterentwicklung – kontinuierlicher Vorgang
  ↳ Software-Evolution

Automatisierte Prozess-Unterstützung

- Computer Aided Software Engineering („CASE")
  - Wird benutzt um Software-Prozesse zu unterstützen
    - Anforderungsanalyse
    - Entwurf + Programmierung
    - Integration + Test
  - Für die meisten Routine-Aktivitäten verfügbar
  - Verbesserung der Produktivität + Qualität (ca. 40%)
  - Einschränkende Faktoren
    - Entwurfsaktivität – gesteuert durch Kreativität (Automatisierung der Routine-Aufgaben hilft hier nicht)
    - Software-Engineering ist Team-Arbeit, erfordert viel Kooperation + Kommunikation (kaum Unterstützung durch CASE-Tools)

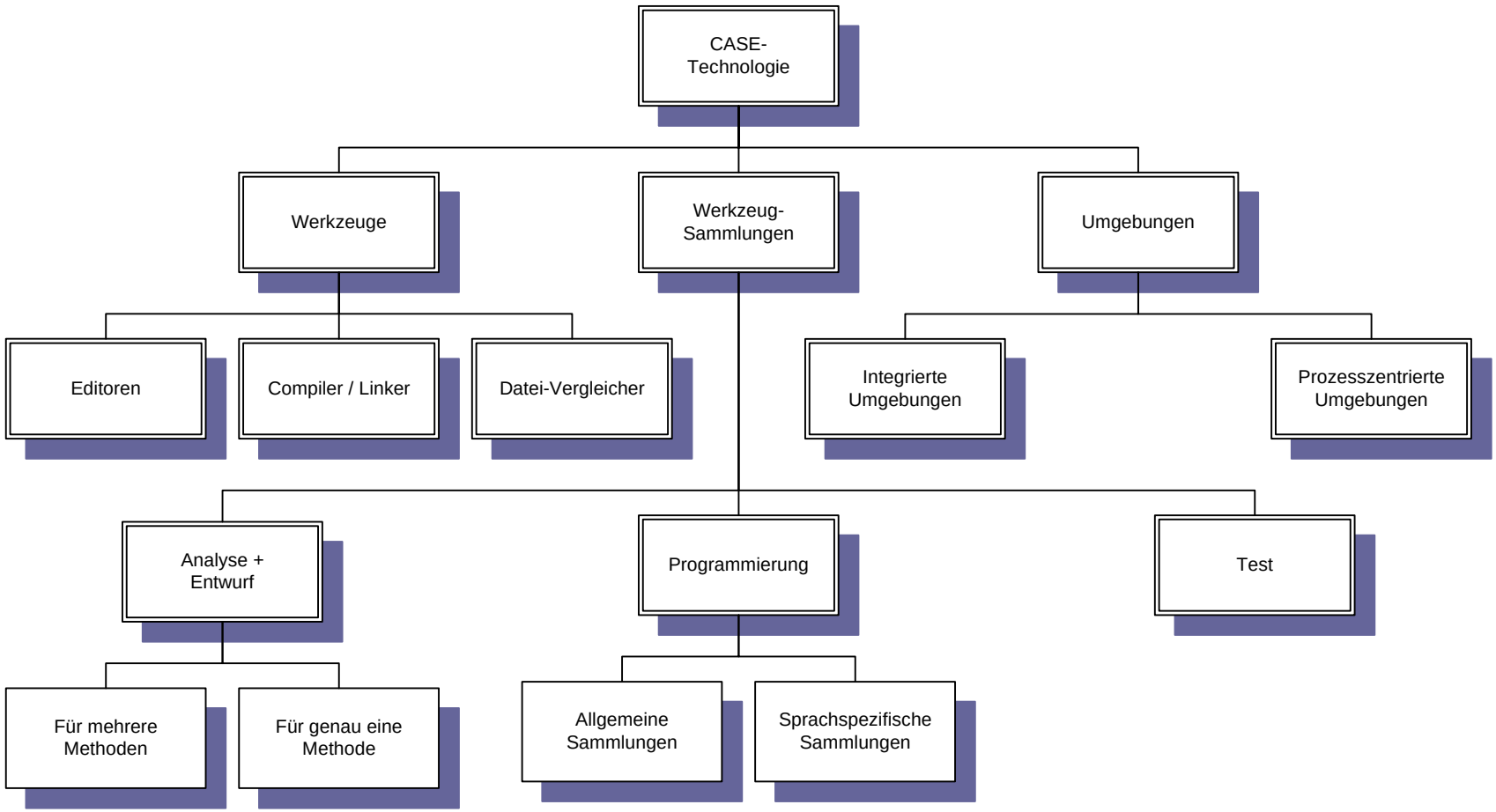
CASE-Klassifizierung

Werkzeugtyp	*Beispiele*
Planungswerkzeuge	Netzplan-Werkzeug, Werkzeug zur Kosten-/Aufwandsschätzung, Tabellen-Kalkulator
Editoren	Text-Editoren, Diagramm-Editoren, Textverarbeitungsprogramme
Werkzeuge für Änderungs-Management	Werkzeuge zur Verfolgung der Anforderungen, Änderungskontroll-Systeme
Werkzeuge für Konfigurations-Management	Versions-Kontroll-Systeme Werkzeuge zur System-Erstellung
Werkzeuge zum (Rapid-)Prototyping	Abstrakte Spezifikationssprachen Generatoren für Bediener-Oberfläche
Werkzeuge zur Methoden-Unterstützung	Entwurfs-Editoren, Data-Dictionaries, Codegeneratoren, 4GL-Werkzeuge

CASE-Klassifizierung

Werkzeugtyp	*Beispiele*
Sprachverarbeitende Werkzeuge	Compiler, Interpreter
Werkzeuge zur Programm-Analyse	Generatoren für Querverweise, statische + dynamische Analyse-Programme
Test-Werkzeuge	Generatoren für Testdaten, Test-Simulatoren Programme zum Vergleich von Dateien
Werkzeuge zur Fehlerbehebung	Interaktiver Debugger
Dokumentations-Werkzeuge	Seiten-Layout-Programme Bildbearbeitungs-Programme
Re-Engineering-Werkzeuge	Querverweis-Systeme, Systeme zur Neustrukturierung von Programmen

CASE-Werkzeuge - Typisierung



Zusammenfassung – Software-Prozesse

- Aktivitäten zur Software-Herstellung
- Softwareprozesse bestehen aus den Phasen
  - Spezifikation
  - Entwurf
  - Implementierung
  - Validierung
  - Weiterentwicklung
- Vorgehensmodelle beschreiben den Aufbau der Software-Prozesse (z.B. Wasserfall-Modell)
- Iterative Vorgehensmodelle – Zyklus von Aktivitäten
- Anforderungs-Analyse zur Entwicklung der Spezifikation
- Entwurf + Implementierung = Umwandlung in Software
- Software-Validierung = Überprüfung der Ergebnisse
- Weiterentwicklung ist die Evolution von Software
- CASE-Tools zur Unterstützung der einzelnen Phasen

Literatur – Software Engineering

- Skript Informatik II — Prof. Dr. Kühn / Fb W FH Karlsruhe
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Skript Software Engineering — Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/se_I/
- Skript Software Engineering II — Bernd Kahlbrandt FH Hamburg
  http://www.informatik.fh-hamburg.de/~khb/st4se2/
- Software Engineering
  Ian Sommerville — (ISBN3-82737-001-9)
- Software Engineering
  - Grundkurs für Praktiker –
  Roger S. Pressman — (ISBN 3-89028-163-X)
- Software Entwurf
  - Methoden und Werkzeuge –
  A. Schulz — (ISBN 3-486-21608-2)
- Software Engineering und Prototyping
  Thorsten Spitta — (ISBN 3-540-17542-3)
- CASE
  Helmut Balzert — (ISBN 3-411-03224-3)
- Software-Qualitätssicherung
  Ernest Wallmüller — (ISBN 3-446-15846-4)

Software Engineering

Informatik II.

3. Software-Entwicklung

– Projekt-Management –