2. Aktivitäten und Dokumente im Lebenszyklus

Ziele:

Klärung des Softwareentwicklungsprozesses und seiner Endprodukte auf Lebenszyklusebene

Vorstellung der Variantenvielfallt von Lebenszyklusmodellen

Abgrenzung des ingenieurmäßigen Vorgehens von Alternativen

Stand: 15.09.2008

Aktivitäten/Dokumente für das Phasenmodell

Anforderungsermittlung/-festlegung (vgl. Kap. 4, 5)

- Problemanalyse (problem analysis, system analysis):

  Problem und seine Umgebungsbedingungen vollständig, eindeutig und präzise beschreiben, insbesondere

  - Bedienerprofil (Art, Anzahl, Eigenschaften der Bediener)
  - Systemumgebung (Hardware, vorgefundene Software)
  - Funktionalität des neuen Systems auf unterschiedlichen Ebenen
  - weitere Parameter, z.B. Effizienz, Schutz, Sicherheit
  - Bedieneroberfläche
  - Verzahnung mit Anwendungssystem

- Ergebnis ist Anforderungsdefinition (requirements definition, requirements specification):
  - Allgemeines: Zweck, Bedienerprofil, Systemumgebung
  - gewünschte Funktionen
  - korrekte/falsche Eingaben und zugehörige Ausgaben/ Systemreaktionen
  - Bedieneroberflächengestaltung

  } → Bediener-dokumentation

  - Festlegung Effizienzparameter
  - Festlegung Testfälle

  } → QS-Dokumente

  - Festlegung Schutz-, Sicherheitsaspekte
  - Dokumentationsanforderungen
  - Qualitätssicherungsanforderungen

- Durchführbarkeitsstudie (zu Problemanalyse gerechnet):
  - technische (überhaupt, unter geg. Bed.), personelle, ökonomische

  } → Durchführbarkeitsbericht

  - Zeitplan, Personalaufwand, Kosten/Nutzen, Risiken

- Unternehmerische Entscheidung über die Durchführung oder Revision der Aufgabenstellung } → Vertrag

- Bereits bei Erstellung der Anforderungsdefinition Diskussion über zukünftige Erweiterungen, z.B. durch Brainstorming

Bauplanerstellung (vgl. Kap. 6, 7)

- Entwurf (Design, Architekturmodellierung, Progr. im Gr.):
  - Modell des Innenlebens des Systems auf grober Ebene:
    Zerlegung in Module (Exporte)
    Festlegung der Beziehungen (Importe)
  - unter statischen Gesichtspunkten (was, nicht wie)
  - verträglich mit Anforderungsdefinition und implementierbar
- Ergebnis ist Entwurfsspezifikation (Spezifikation, Bauplan, Softwarearchitektur):
  - ist statische Festlegung
  - in einer Sprache fixiert (formal, semiformal, informell)
  - Grundlage für Überprüfung gegen Anforderungsdefinition
  - Grundlage für Überprüfung der Implementierung einzelner Module und ihrer Integration
- Handicap: nimmt breiten Raum ein, erst spät Codezeilen
  - Nutzen erst längerfristig: z.B. Wartbarkeit
  - Voraussetzung für die Gewinnung von Strukturwissen über Anwendungsbereich, Klasse von Systemen etc.

Programmierung/Codierung

- Implementierung (Progr. im Kleinen):
  - Bausteine ausprogrammieren (in der Regel von verschiedenen Personen)
  - ausgehend von entsprechendem Teil der Entwurfsspezifikation (Export und Import)
  - heißt Daten- und Ablaufstrukturen festlegen
  - bei niedriger Programmiersprache (FORTRAN, Assembler): abstrakte Implementation in Pseudocode
  - Modultest
- Ergebnis sind ausgetestete, einzelne Modulimplementationen:
  - leicht verständlich, überprüfbar, änderbar, nicht optimiert
  - dynamisches Zusammenspiel der einzelnen Module noch nicht gesichert
- Wird in der Vorlesung nicht weiter angesprochen, Grundausbildung sollte vorliegen:
  - Programmiertechnik: Wie geht man vor?
  - Programmiersprachenkenntnisse: Datentypkonstruktoren, Kontrollstrukturen, Strukturierung von Ausdrücken
  - Korrektheit: part. Korrektheit, Termination
  - Modultest teilw. (vgl. Kap. QS)
  - Komplexität (Messen, Rechnen)

Integration

- Integration/Funktions-/Leistungsüberprüfung:
  Idealfall:
  - Entwurfsspezifikation gegen Anforderungsdefinition konsistent
  - Korrektheit jedes Moduls bewiesen (verifiziert)
  - Integration, Funktionsüberprüfung größtenteils fertig

  Praxis:
  - Integration von Modulen zu größeren Einheiten
  - diese wieder zu größeren Einheiten bis Gesamtsystem
  - Überprüfung durch Integrationstest
  - entdeckt dabei Implementierungs-, Entwurfs- und Problemanalysefehler
  - Leistungsmessung nach funktionaler ”Korrektheit”
  - gegebenenfalls Optimierungen, um Leistungsparameter der Anforderungsdefinition zu erfüllen
- Ergebnis: lauffähiges, überprüftes, ”dokumentiertes” Softwaresystem

Übergabe

- Installation und Abnahme:
  - Übertragung eines Softwaresystems in seine reale Umgebung (eventuell andere Hardware)
  - Installationstest
  - Abnahme durch Auftraggeber
- Ergebnis: einsatzfähiges Softwaresystem

Veränderung

- Wartung (Pflege):
  - Bedeutung (vergleiche Graphik Boehm 60% /Boe 76/)
  - Begründung: ungenaue Anforderungen, keine saubere Softwarearchitektur, zu wenig über Änderungen nachgedacht, Implementierungsfehler: Wartung durch Erker
- Ergebnis: verändertes Softwaresystem
  - nach einigen Änderungen keine klare Struktur mehr (Spaghettiteller)

Ende des Betriebs

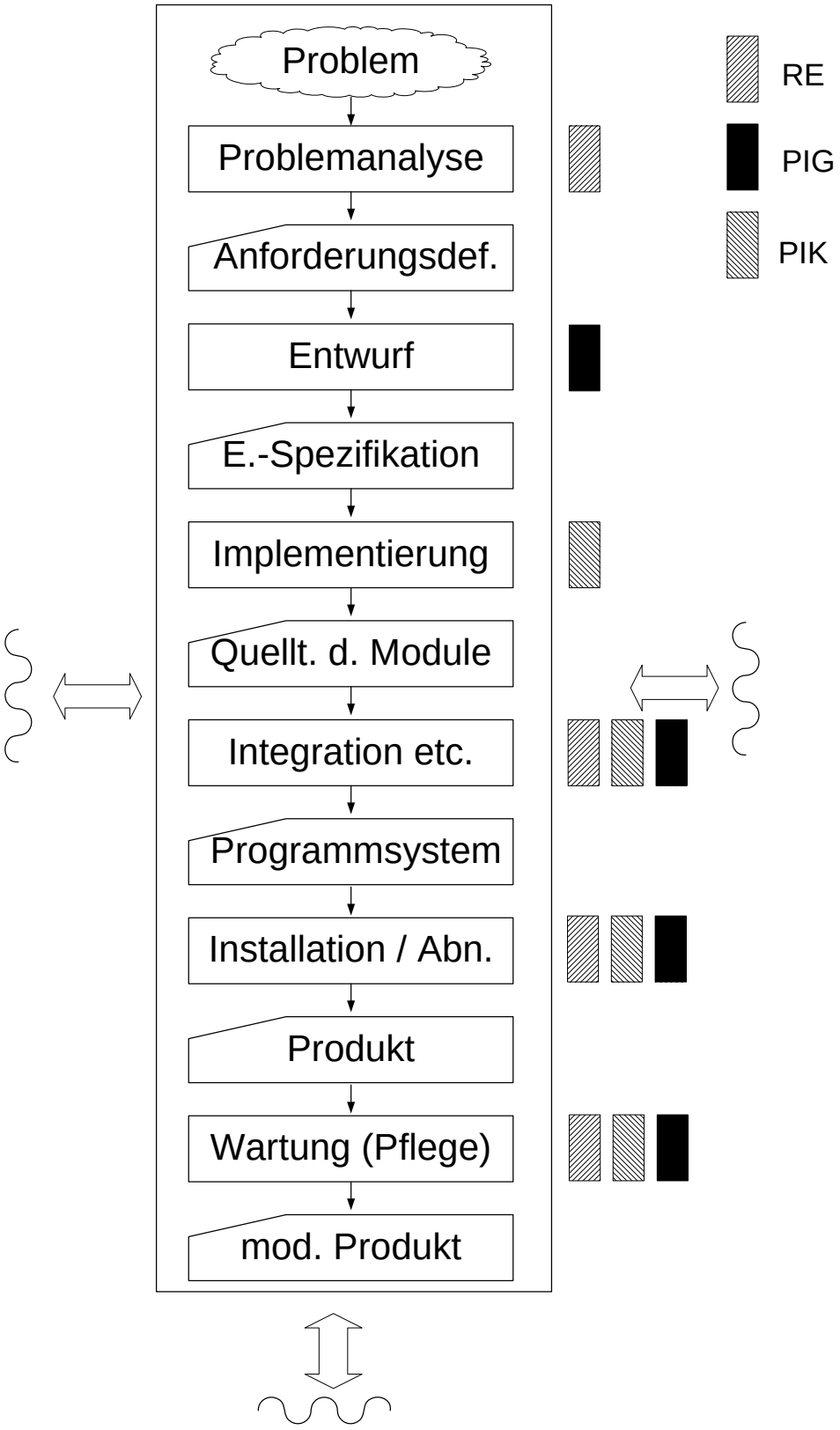
- bewußte Zurückziehung eines SW-Systems

Arbeitsbereiche und deren Zusammenhang

Zusammenfassung der Aktivitäten in Arbeitsbereiche

- Phasenmodelle
  - zeitlicher Verlauf des Entwicklungs-/Wartungsprozesses
  - Abhängigkeit bezüglich ”ist Ergebnis von”/”wird benötigt für”/”kommt vor” für Entwicklungsphasen, ”kommt vor” für Wartungsphase
- Einteilung in Arbeitsbereiche, Motivation:
  - wo wird auf gleichem logischen Niveau modelliert
  - Zusammenfassung von Tätigkeiten, die zeitlich verstreut liegen
  - Aufgabe der Unterscheidung zwischen Erstellung und Modifikation (Erstellung/Wartung ist dauernde Modifikation)
  - Voraussetzung für Klärung von Zusammenhängen in und zwischen Arbeitsbereichen
  - Entwicklung entsprechend abgestimmter Hilfsmittel für arbeitsbereichsinterne sowie übergreifende Teilentwicklungsprozesse

- Wo tauchen Tätigkeiten dieser Arbeitsbereiche im Phasenmodell auf?



- drei technische Arbeitsbereiche, zunächst grob:
  - Definieren/Verändern der Anforderungen: Außenverhalten des Systems
    Angewandte Hilfsmittel: Anforderungstechnik (Requirements Engineering)

  - Programmieren im Großen (Entwurf, Bauplanerstellung, Architekturmodellierung): Festlegung/ Veränderungen der Struktur des Systems auf grober Ebene gemäß Anforderungsdefinition
    Angewandte Hilfsmittel: Entwurfstechnik

  - Programmieren im Kleinen: Realisierung/Veränderung einzelner Module
    Angewandte Hilfsmittel: Programmiertechnik

- Definition des Begriffs Programmieren im Großen

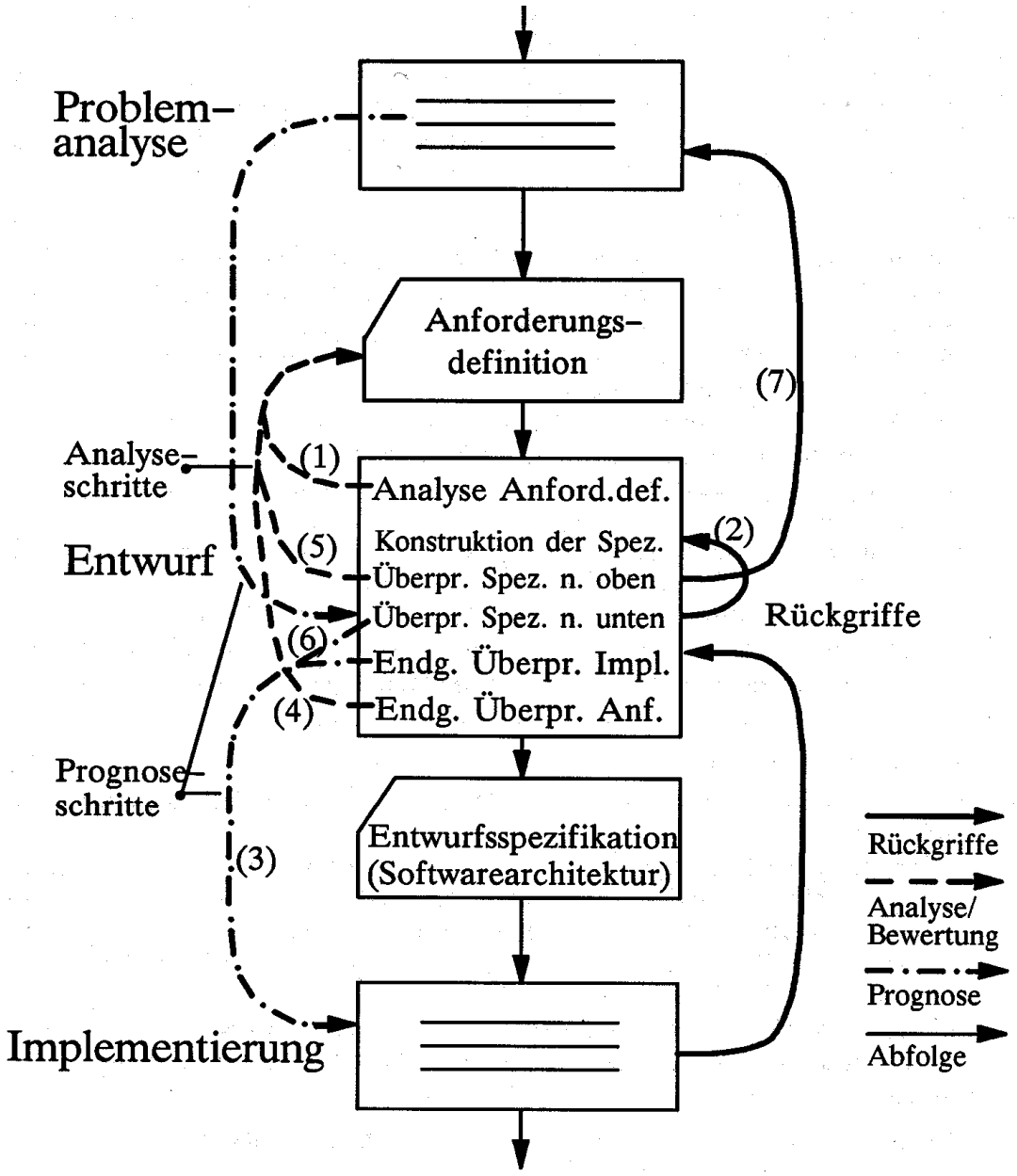


Fig. 1.5: Einzelaktivitäten (Feinstruktur) der Entwurfsphase

aus /Nag 90/

- jede Lebenszyklusphase:
  - Analyse
  - inkrementelle Konstruktion, Analyse, Überprüfung, Prognose
  - “abschließende” Überprüfung nach oben
  - “abschl.” Überprüfung nach unten
  - “abschl.” Übergabe

- bisher nicht berücksichtigt (gültig für jede Phase):
  - untersch. Tätigkeiten
  - untersch. Ergebnisse
  - Zusammenhang

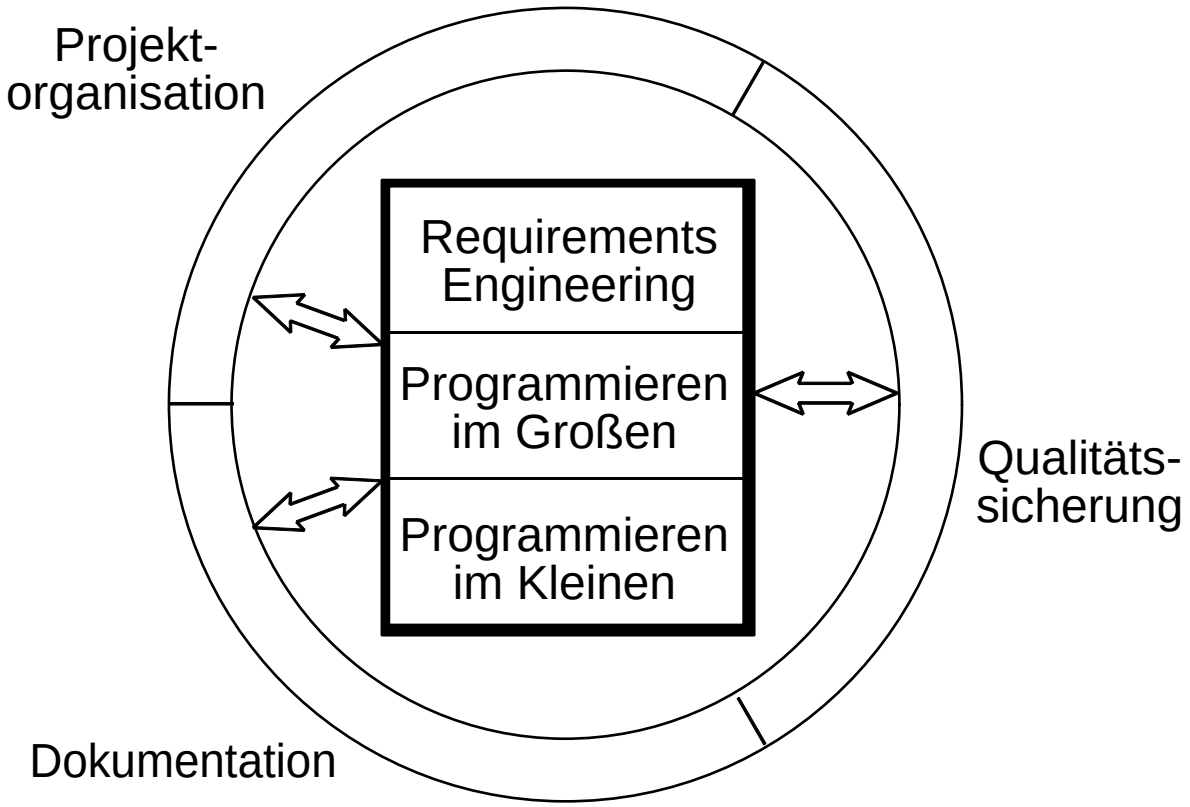
  } Teilkonfigurationen

- Definitionen:
  Requirements Engineering: ...
  Programmieren im Großen:
  - Analyse der Anforderungsdefinition unter Aspekten des Entwurfs
  - stückweiser Entwurf eines Softwaresystems aus Modulen und Teilsystemen
  - stückweise Überprüfung der entstehenden Entwurfsspezifikationskomponenten: intern, gegen die Anforderungsdefinition und auf Implementierbarkeit, Integrierbarkeit sowie Wartbarkeit
  - abschließende Überprüfung gegen die Anforderungsdefinition
  - abschließende Überprüfung auf Realisierbarkeit
  - Übertragung der Entwurfsspezifikation in eine Programmiersprache: ”Codieren im Großen”
  - Heranziehen der Architektur zu Integration und Funktionsüberprüfung von Modulen und Teilsystemen
  - Heranziehen der Architektur zu Leistungsüberprüfung von Modulen, Teilsystemen des Gesamtsystems
  - Heranziehen der Architektur zu Installation des Gesamtsystems aus den Komponenten der Architektur
  - Veränderung der Architektur bei Rückgriffen, insbesondere in der Wartung, durch Neuangehen aller obigen Schritte

  Programmieren im Kleinen: ...

- weiterer Arbeitsbereich Dokumentation (s. späteres Kap.)
  - Entwicklungsdokumentation (technische Dokumentation): Entscheidungen begründen, Struktur erklären
  - Bedienerdokumentation
  - Projekthandbuch: invariante Teile des Projekts
- weiterer Arbeitsbereich Qualitätssicherung (s. späteres Kap.), z.B.
  - formal (Verifikation)
  - experimentell (Test)
  - manuell (Review, Inspektion, Walkthrough)
- weiterer Arbeitsbereich Projektorganisation (s. späteres Kap.)
  - Einteilung 1:
    Projektplanung
    Projektführung (Projektmanagement)
    Projektüberwachung
    Projektadministration
  - Einteilung 2:
    Management of the Process
    Management of the Products
    Management of the Project
  - Einteilung 3 (s.u.)

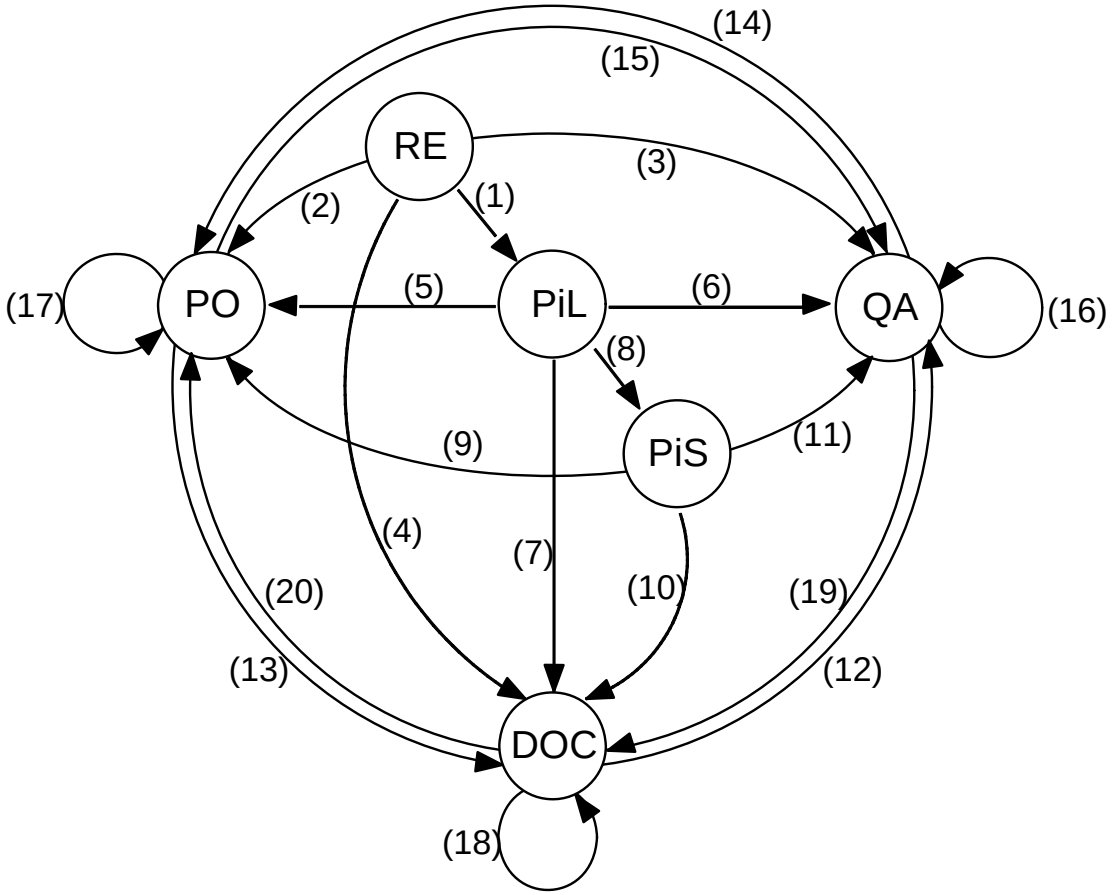
- Einteilung in Arbeitsbereiche



aus /Nag 90/

- Rollenzuteilung für Aufgaben
- Verzahnung der Arbeitsbereiche (s.u.)
- mehrdeutige Begriffe der Softwaretechnik:
  - Spezifikation
  - Implementierung
  - Realisierung

- Abhängigkeitsbeziehungen zwischen Arbeitsbereichen



aus /Nag 96/

1) Architecture document(s) have to be consistent with the requirements specification documents, e.g. with the functional and nonfunctional description of the system.
2) Document(s) for project planning (black box estimation) depend on the requirements specification; the activities of requirements engineering are identified as activities to be planned, managed, supervised, and replanned on PO level.
3) The requirements specification has to be reviewed, or checked/executed by tools (if the description is executable).
4) The functional and nonfunctional part of the requirements specification has to be documented in the technical documentation and both influence the user documentation.
5) Project organization documents for cost planning (white box estimation) but especially for management and supervision are highly dependent on the architecture. In the architecture, nearly all work packages of the project can

be identified for project organization. Furthermore, architecture modelling itself has to be organized.

6) The architecture is reviewed or checked giving rise to dependency relations between architecture documents and review protocols, check lists etc. Furthermore, the structure of module and integration test documents (test data collection, test drivers/stubs, test order determination plan etc.) is determined from the architecture especially when black-box test is used.
7) The architecture determines most of the technical documentation not only from its structure but also from its contents.
8) The modules to be implemented are determined by the decisions made during architecture modelling. Only the implementation is variable and still there the resources used for a module body are fixed in the architecture.
9) The process of implementing the modules has nearly no influence on the structure of PO documents but on its values (time needed, persons assigned etc.), as the corresponding units are already identified on architecture level.
10) The implementation ideas are described in the technical documentation.
11) The modules are tested according to available test methods and tools. In the case of black box tests the structure of module bodies has rather little influence on quality assurance documents. In the case of white box tests we have a strong correspondence between the structure of module bodies and module test documents.
12) The documentation (user documentation, technical documentation, project handbook) has to be reviewed carefully.
13) Documentation as an activity has to be organized. The corresponding management documents are very little dependent on the documentation structure.
14) Quality assurance has to be managed.
15) All documents of project planning, project management, and project supervision have to be reviewed and/or checked.
16) Quality assurance, procedures, and strategies to be applied in a project have to be reviewed and evaluated. Furthermore, concrete quality assurance documents (e.g. a module test) may be evaluated by a different quality assurance engineer.
17) Project organization itself has to be organized (planned, managed, supervised).
18) The structure of documentation has to be documented.
19) Quality assurance activities have to be documented.
20) Project organization activities have to be documented.

Diskussion Lebenszyklusmodelle

Phasenmodelle sind eine gewollte Vereinfachung

- Rückgriffe
  auf vorangegangene Phasen, dabei Modifikation der entsprechenden Softwaredokumente

  unvermeidbar: durch sorgfältige Modellierung Anzahl und Weite verminderbar (s.u.)

- Beispiele für Rückgriffe:
  - eine Phase zurück
  - mehrere Phasen zurück
- bisheriges Phasenmodell vereinfacht:
  - Phasen keine monolithischen Blöcke (s.o.)
  - Rückgriffe
  - Vorgriffe (Prognose)
  - Wartung ist keine ”Phase” (zeitlich ja, logisch nein)
  - praxisnahes Modell zu kompliziert als Gesprächsgrundlage

- Rückgriffe im Entwicklungsprozess
  - Lebenszyklus: anfangs Problemwolke, am Ende präzise Anweisung für Basismaschine: schwierige Aufgabe!
  - Lebenszyklus fast nie linear!
  - sorgfältige Überlegungen: weniger Rückgriffe
  - sorgfältige Überlegungen: Rückgriffe weniger weit zurück, etwa nur auf vorangegangene Phase
  - Kosten steigen dramatisch mit der Weite des Rückgriffs und der Anzahl der Rückgriffe (insbesondere wenn Software schon im Einsatz, s. nächstes Kap.)

- Beispiele für Rückgriffe:
  - auf Problemanalyse:

    im Entwurf werden notwendige Eigenschaften der Basismaschine entdeckt

    im Entwurf oder in der Implementierung werden unvollständige oder mißverständliche Anforderungen entdeckt
  - auf Entwurf:

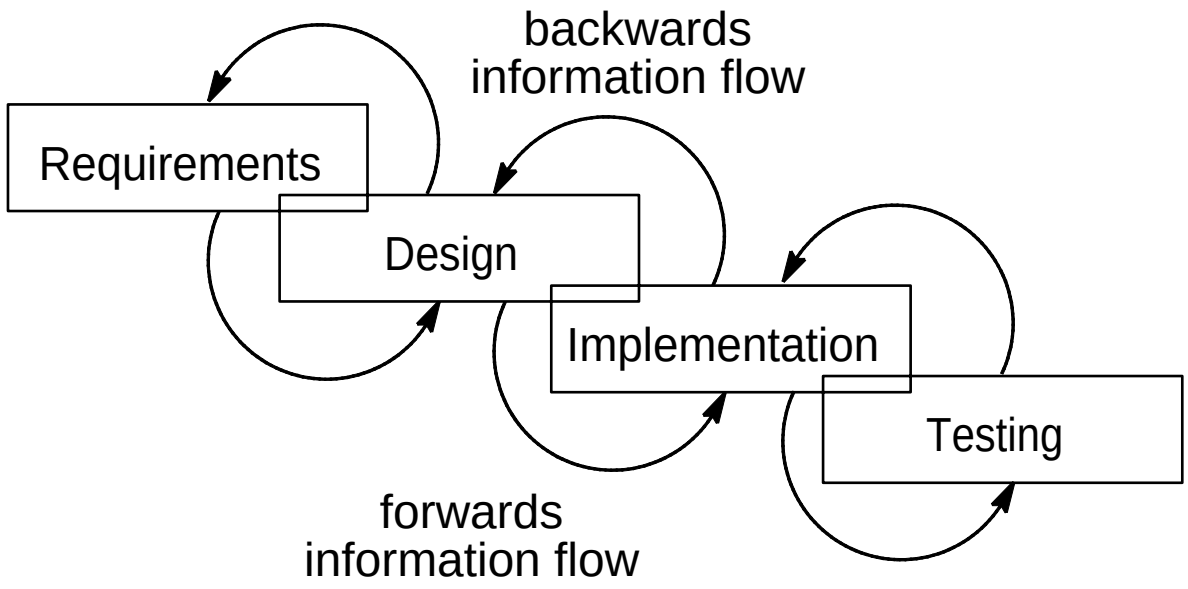
    für Implementierung eines Moduls Informationen nötig, die nicht in der Schnittstelle steht

    für Implementierung ist Information in der Entwurfsspezifikation zu unpräzise
  - von Funktions-/Leistungsüberprüfung nach oben:

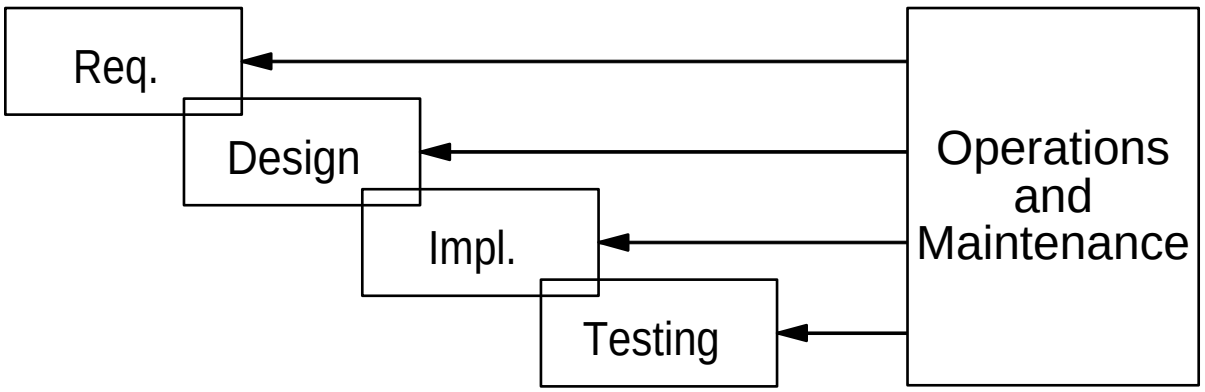
    ...
  - von Wartung nach oben:

    ...

- Unterscheidung Entwicklungs- und Lebenszyklus



The software development cycle



The software life cycle

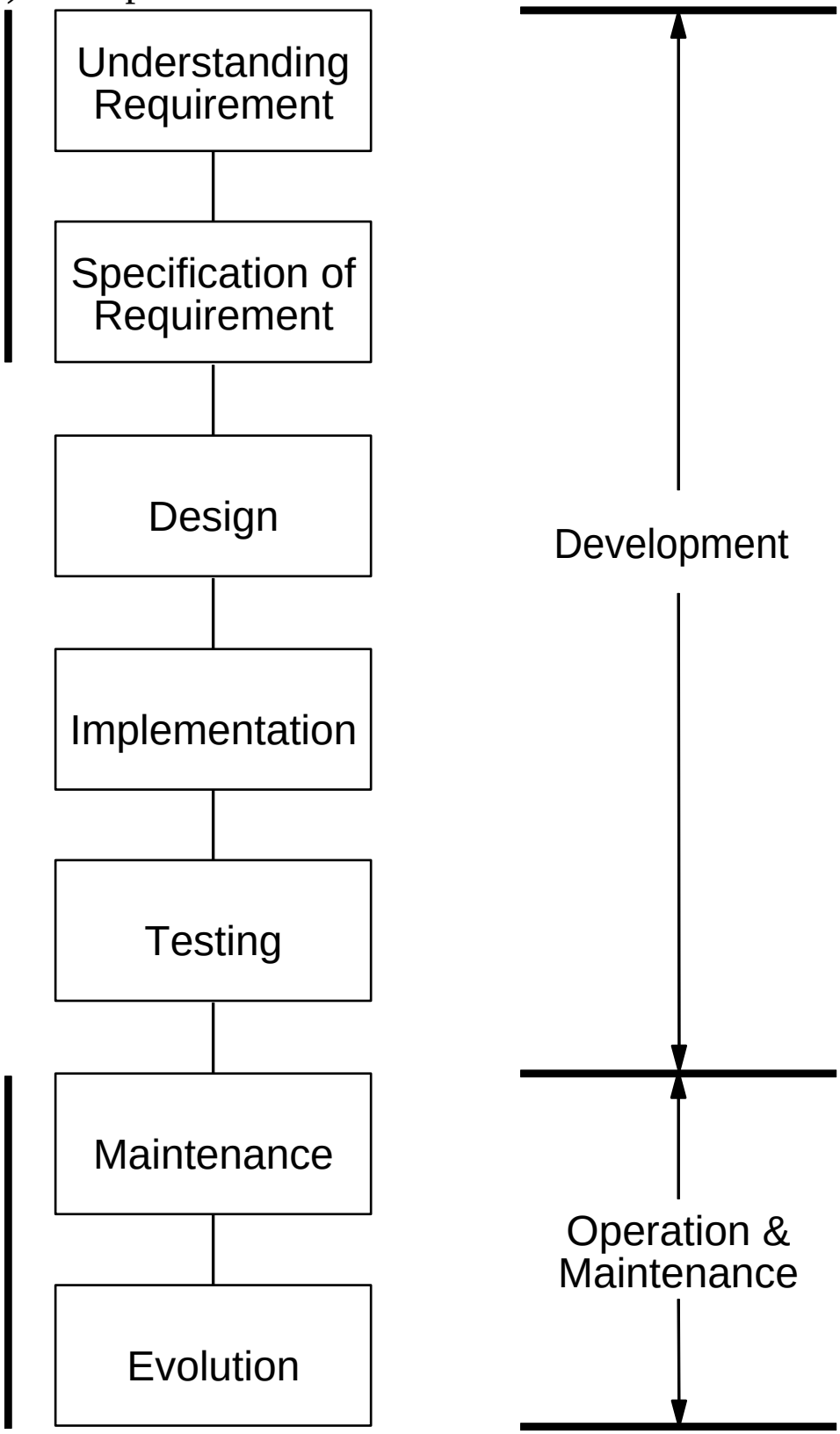
aus /Som 92/

Phasenmodellvarianten

- Varianten
  a) Auftrennen oder Zusammenlegen von Phasen
  b) eingebettete Systeme: erweitertes Phasenmodell für Gesamtsystem
  c) Rapid Prototyping (später i.a. weggeworfen)
  d) partizipative Modelle:

     Bediener arbeiten bei Anforderungsdefinition mit
     Rapid Prototype führt zu Anregungen/Einsprüchen durch Bediener
     Vorbereitung des organisatorischen Systems für späteren Einsatz des Softwaresystems
     Benutzung der Software erscheint als Phase
  e) Berücksichtigung der Feinstruktur von Phasen (s.u.)

- zu (a): Aufspalten/Verschmelzen



aus C.V. Ramamoorthy: Software Engineering - Problems and Perspectives, Computer 10/84

zu (a): Life cycle reporting milestones

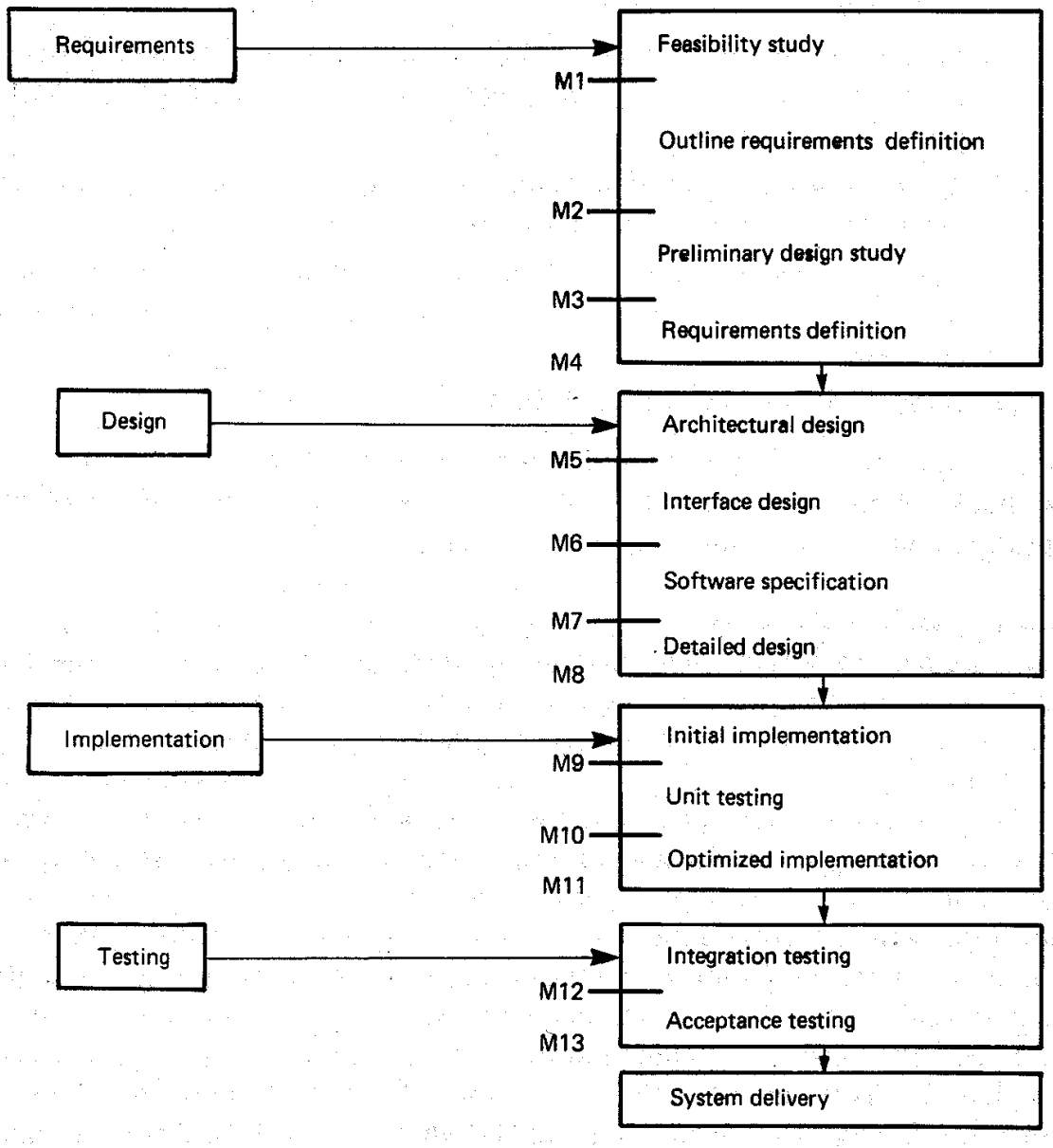
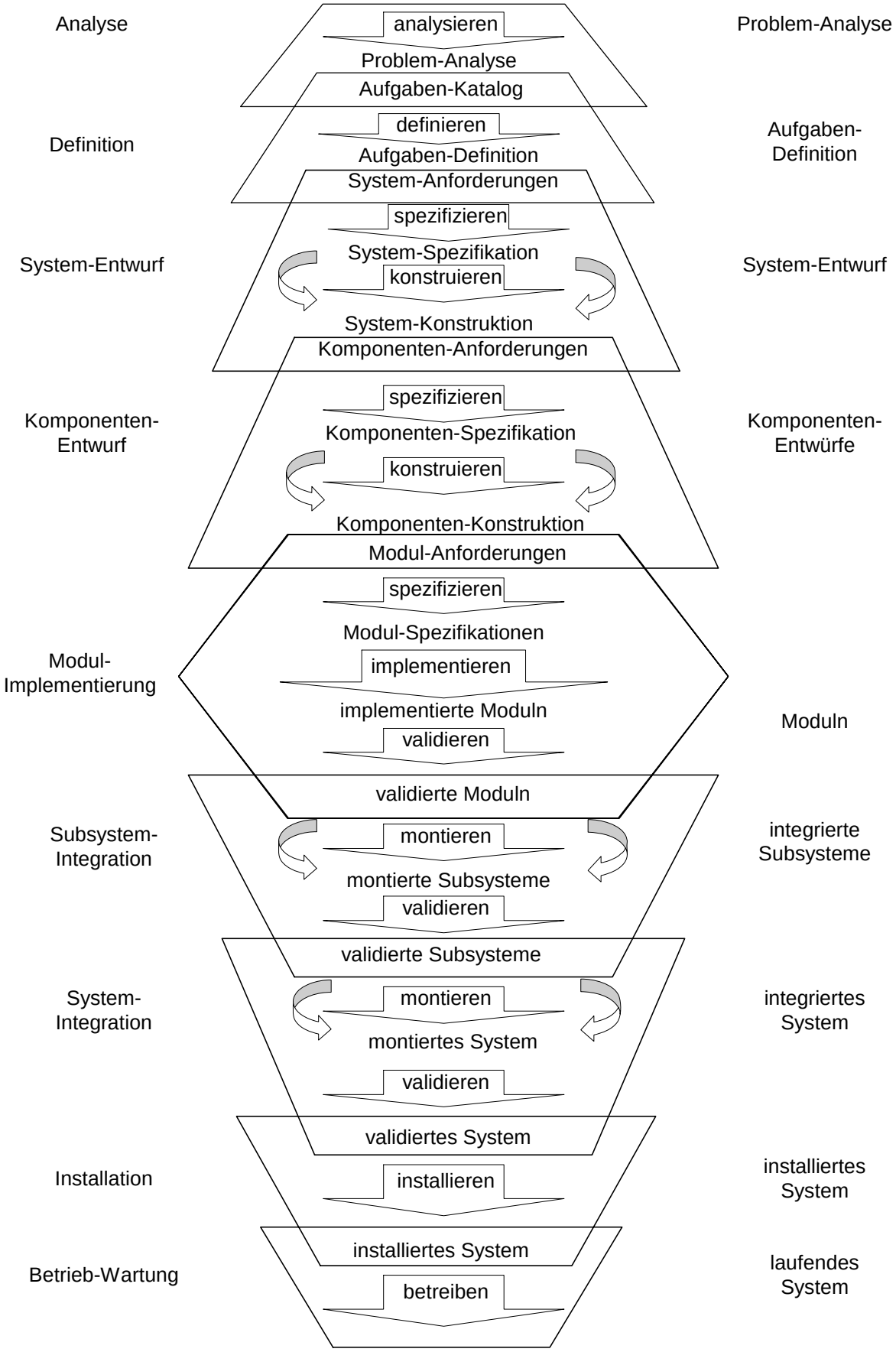


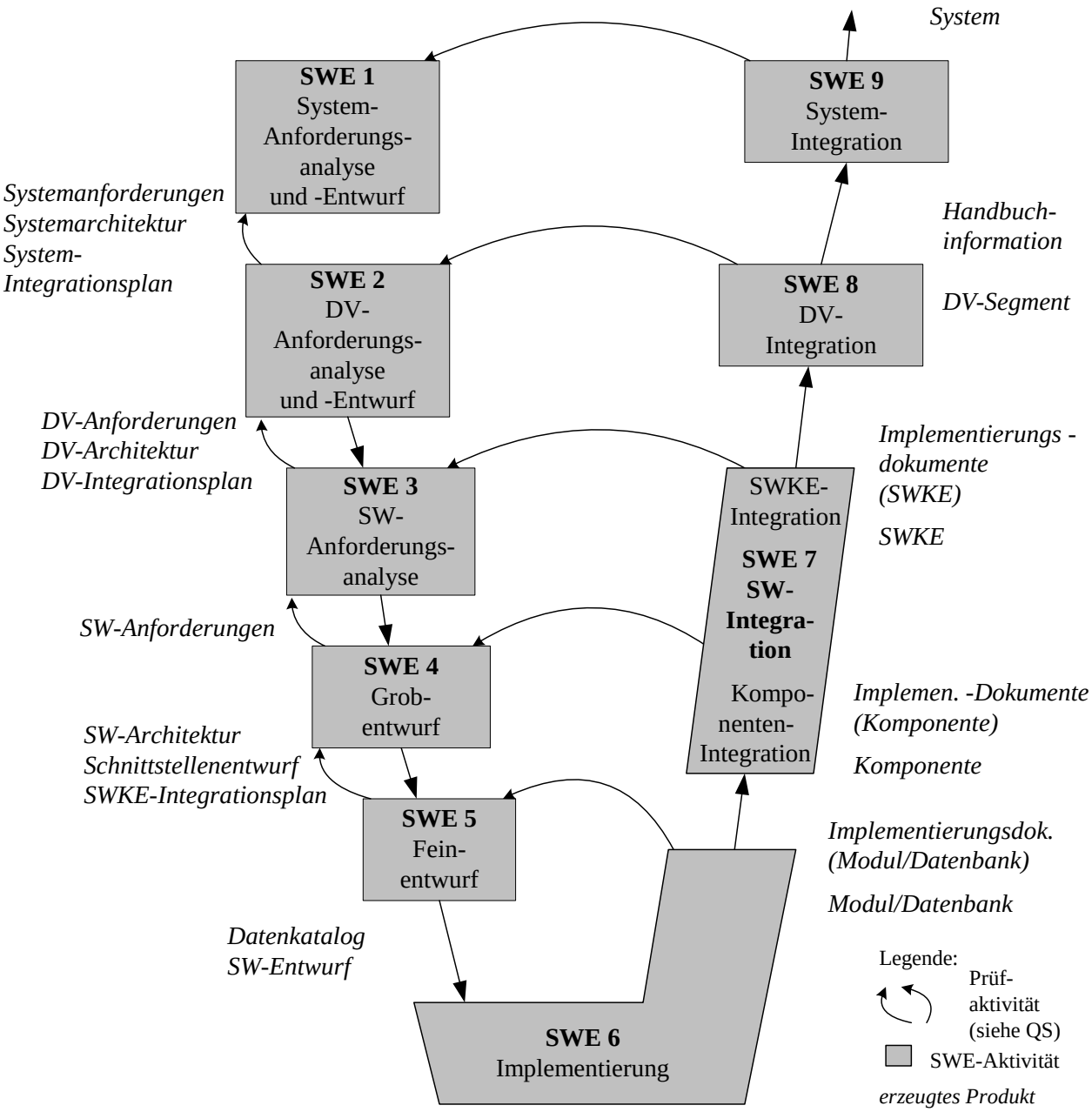
Figure 11.2 Life cycle reporting milestones.

aus /Som 89/, S. 280

zu (a): SOFTLAB-Projektmodell

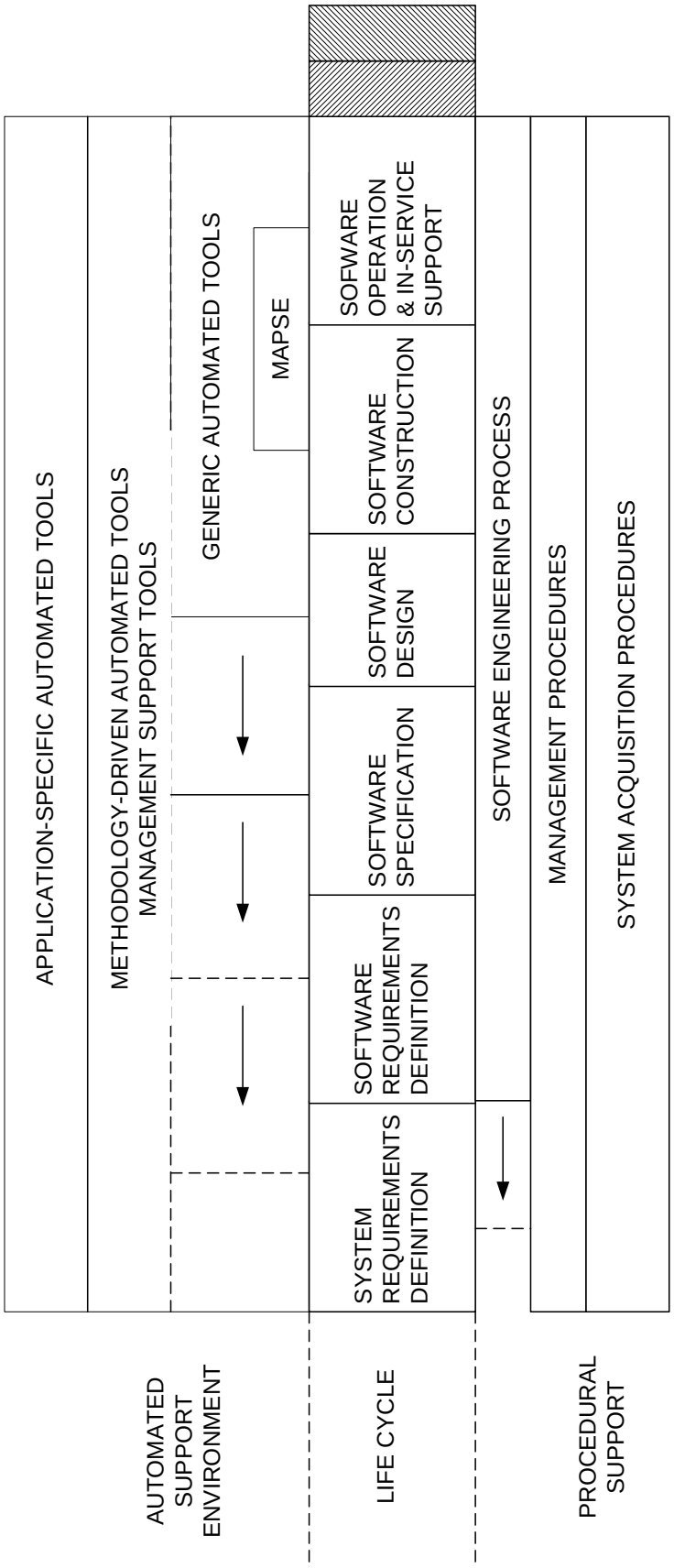


- zu (a und b): Das Submodell SWE



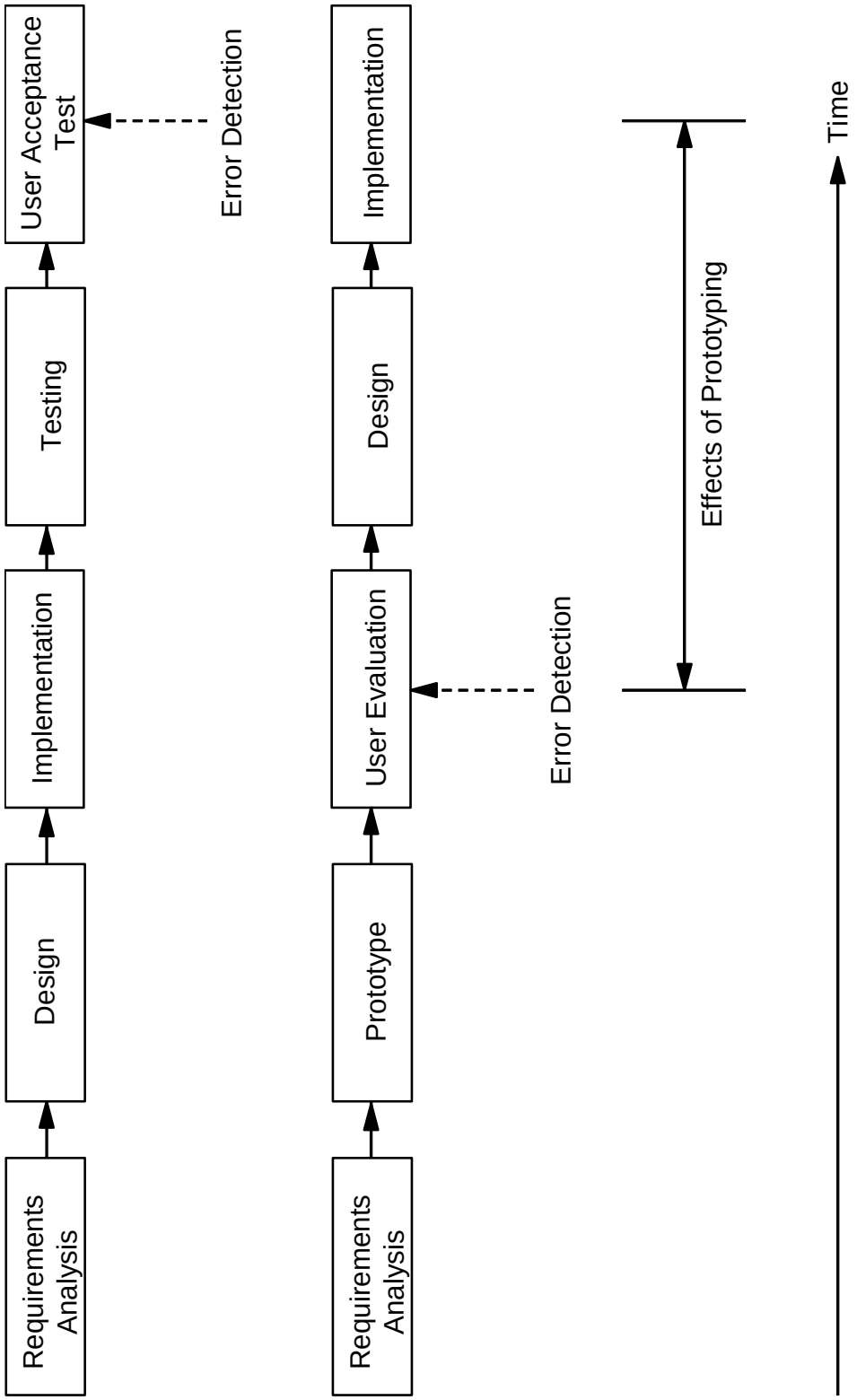
aus /BD 93/

- zu (b): eingebettete Systeme



aus Druffel: The STARS Program ..., Computer 11/83, S. 20

- zu (c): Rapid Prototyping



nach C.V. Ramamoorthy: Software Engineering - Problems and Perspectives, Computer 10/84

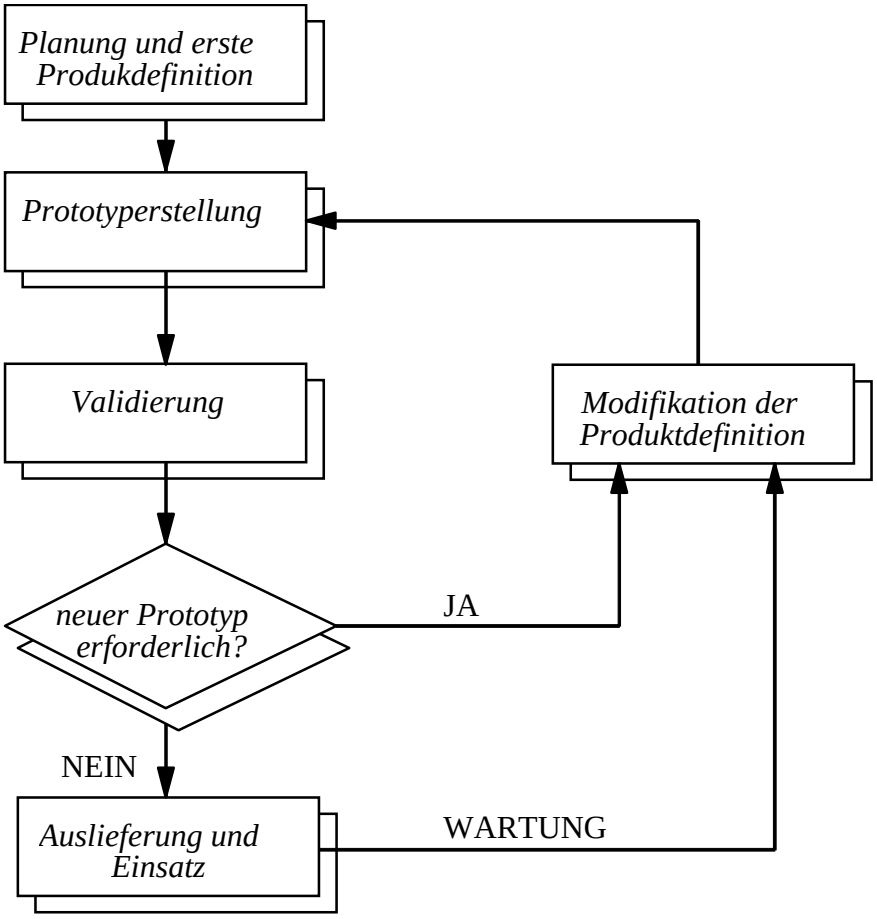
Ansätze für das Rapid Prototyping

- Klassifizierung von Prototyping nach
  - Art der Realisierung des Prototyps:
    direkte Ausführung, Generierung, Zusammenstellung aus vorgefertigten Teilen, Neuimplementierung in entsprechender Sprache
  - Umfang des Prototyps:
    (vollständige) Funktionalität, teilweise Funktionalität oberhalb einer Schicht, nur spezifische Teile
  - Verbindung zum fertigen System:
    Validierung der Wünsche und wegwerfen, Ausgangsbasis für Weiterentwicklung
- Unterschiedliche Prototyping-Ansätze:
  - exploratives Prototyping:
    fachliche Anforderungen klären, Kommunikation mit Auftraggeber, eventuell mehrere Prototypen, Realisierung in beliebiger Weise, in der Regel (vollständige) Funktionalität, Prototyp wird weggeworfen
  - experimentelles Prototyping:
    Nachweis der Tauglichkeit von Realisierungskonzepten, in der Regel Teilbereich realisiert, der ”kritisch” ist, wird auf Anforderungsdefinition aufgesetzt (direkte Ausführung, Generierung etc.), Prototyp als Produkt nicht weiterverwendet
  - evolutionäres Prototyping:
    Endprodukt entsteht aus Folge von ”Prototypen”, Anforderungsdefinition ändert sich außerdem

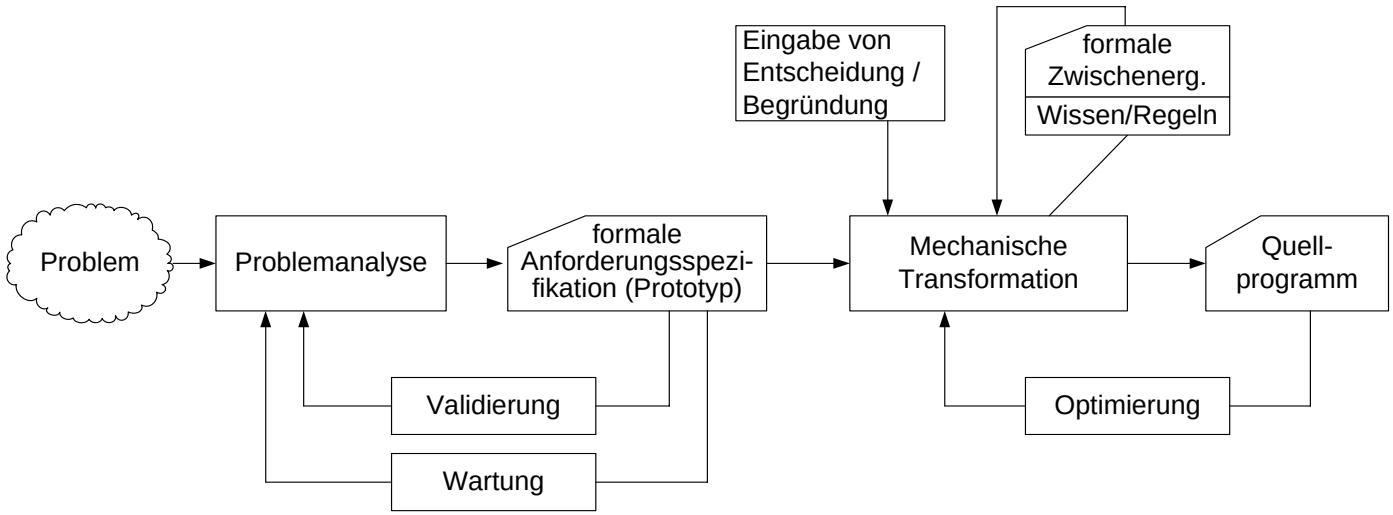
Alternative Ansätze

- Ansätze:
  - evolutionäre Ansätze: SW-Erstellung heißt Prototyp-Erstellung/-Modifikation
  - Transformationsansätze: Entwicklung heißt Anwendung wohldefinierter und überprüfter Transformationen
  - KI-Ansätze: System lernt fortwährend durch Aufbau von Wissen und Regeln

- Evolutionäre Softwareentwicklung



- Ansatz Programmer's Apprentice ist Kombination:
  - Aufheben des Wissens des Entwicklungsprozesses
  - ausführbare Anforderungsspezifikation
  - Automatisierbarkeit des Entwicklungsprozesses



aus Balzer, Cheatham, Green: Software Technologies ..., Computer 11/83

Realisierung:
- automatische Übersetzung
- interaktive Eingabe von Entwurfsentscheidungen
  System führt Transformationen aus
  anfangs einzugeben, später automatisch, da System "lernt"

analog Optimierung

analog Modifikation

- Spiralmodell

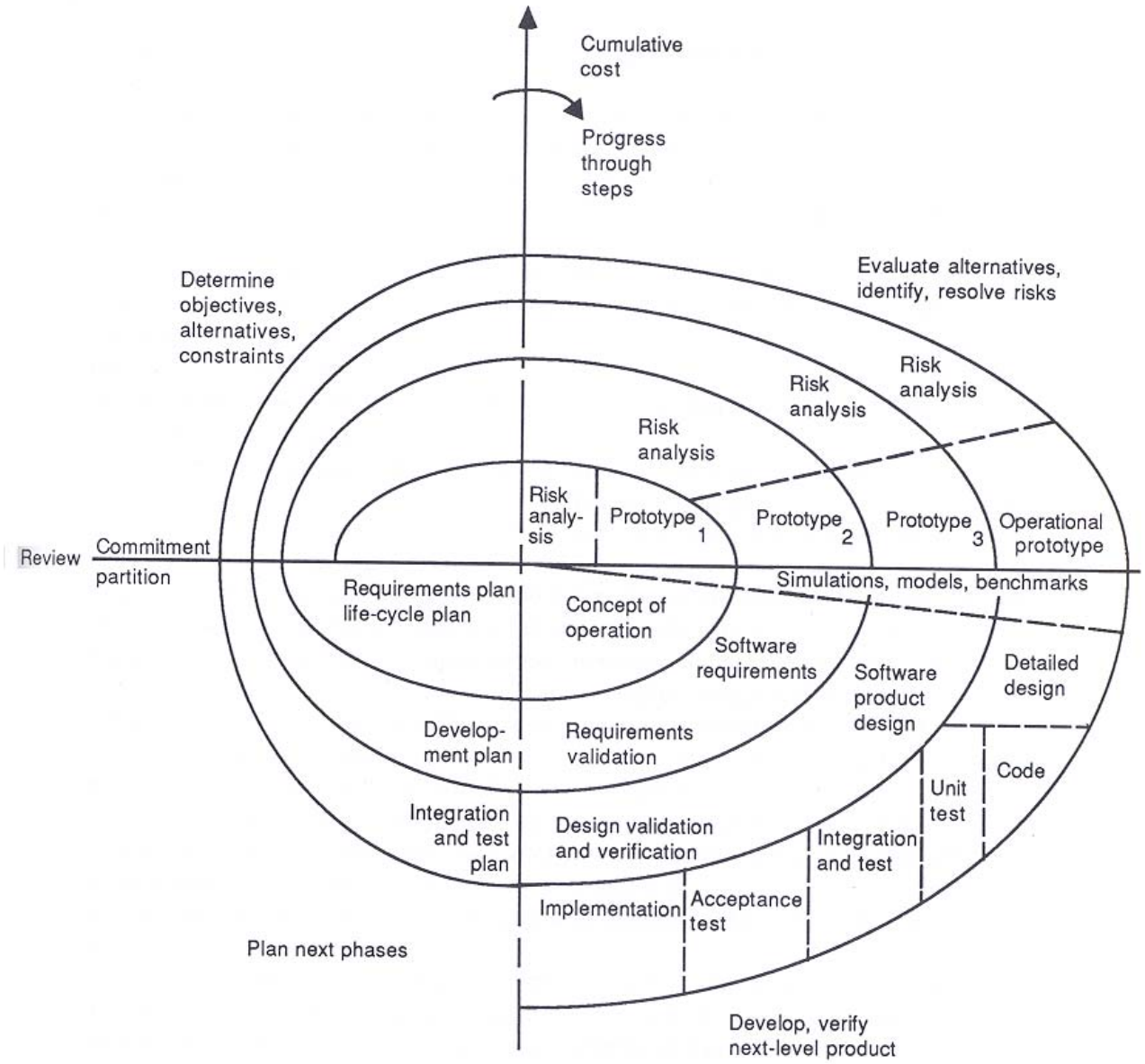
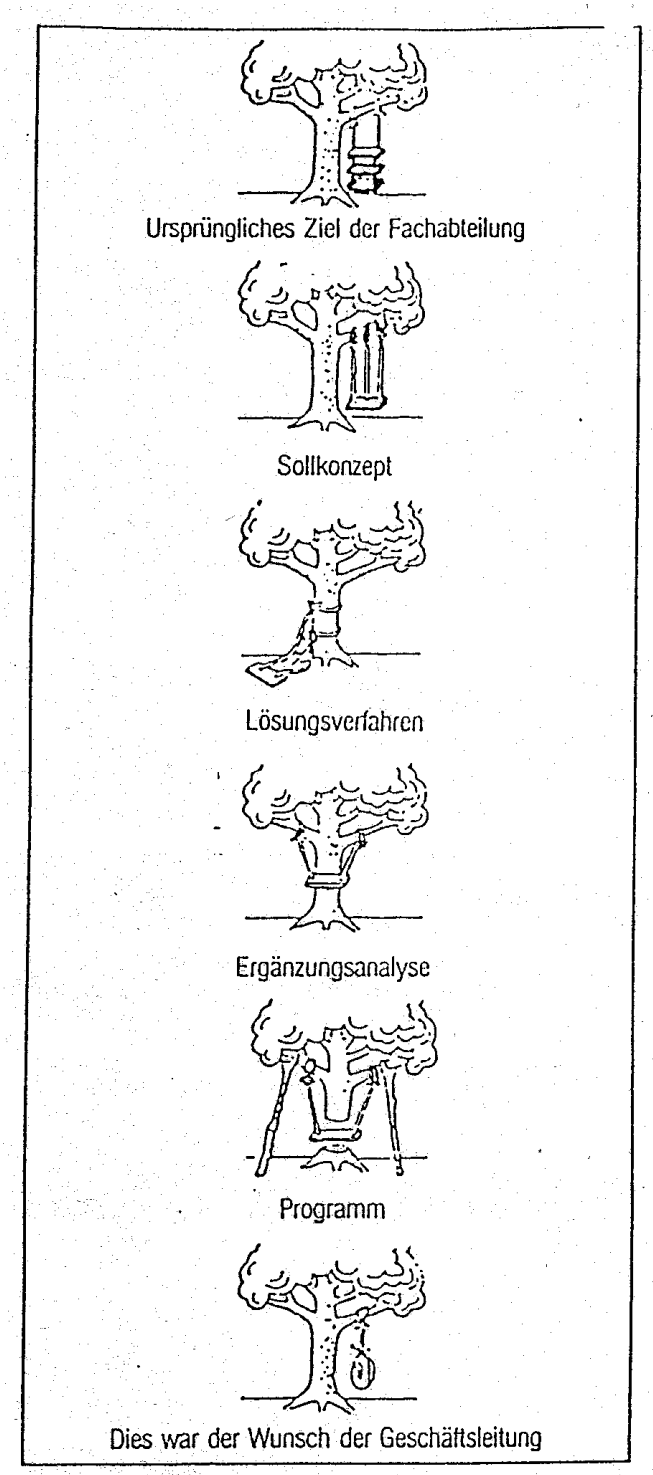


Figure 7.4 The spiral model. (From Boehm [1988], ©1988 IEEE, by permission of IEEE.)

aus /Boehm 88/, vgl. auch /Boe 86/



Planung und Wirklichkeit

Zusammenfassung/Einordnung

Unterscheidung diskret/kontinuierlich

- diskrete Ansätze
  - Phasenmodell (viele Varianten)
  - Arbeitsbereichsmodell
- kontinuierliche Ansätze
  - KI
  - algebraische Spezifikation
  - Transformationsansätze
  - interpreterorientierte Ansätze

Genaueres zur Unterscheidung

discrete apporaches (engineering-like approaches)	continuous approaches (AI-like or semantical approaches)
• different phases or working areas expressing different separate perspectives of a system (requirements specification, architecture etc.)	• one broad ’working area’, no different perspectives are felt to be necessary
• different languages for different document classes usually each incorporating different paradigms (multi-paradigm, multi-language SDEs)	• a broadband language is able to ’handle’ all perspectives with one or more paradigms (mono-language, mutli/mono-paradigm SDE), some working areas are ignored
• overall configurations are structured by dependency relations, overall processes contain different technical subprocesses	• there is one big ’program’ and a continuous process to build or change it (a system is a moving target /1. GR 83/)
• any document has to be elaborated to a certain state (not necessarily complete as in the waterfall model) and should be quality assured before the work on a dependent document can start, transitions between documents being handled by developers and/or tools	• no restriction about how the portions of the big system are elaborated, process steps are delivered by developers or by tools
• a team of different people, having different roles, is developing a system usually in different subprojects	• no restriction of developer freedom
• the software system to be developed is used in the field, i.e. by many people far away of the development context	• developers are modifiers and users, systems are prototypes or systems usually only used in the same or another lab

aus /Nag 96/

Strukturierung bisher auf sehr grobem Niveau

- Lebenszyklusmodellebene in diskreten Modellen (gröbstgranular):
  - ein Arbeitsbereich/eine Phase z.B. für Architekturmodellierung
  - dabei entstehen unterschiedliche Dokumente, z.B. Grobarchitekturdokument, Teilsystemdokumente
  - die entsprechenden Prozesse Grobarchitektur-, Teilsystemarchitekturentwicklung sind nicht erfaßt
- grobgranulare Ebene (Workflow-Ebene):
  - betrachten alle Prozesse/Dokumente, die entstehen
  - benutzen diese Information zur Koordination der unterschiedlichen Entwickler
  - betrachten nicht die Inhalte der Dokumente und der zugehörigen technischen Prozesse
- technische Ebene:
  - Wie wird Teilsystementwicklung durchgeführt (Prozesssicht)?
  - Wie ist das entstandene Ergebnis, Zwischenergebnis strukturiert (Produktsicht)?
  - Wie kann Prozess verbessert werden?
  - Welche Hinweise, Regeln gibt es für das Ergebnis?

Aufgaben zu Kap. 2:

1. Versuchen Sie, gemäß Fig. “Definition des Begriffs PiG”, die Feinstruktur der Problemanalysephase zu formulieren sowie deren Rückgriffe und Vorgriffe, soweit diese vorhanden sind. Welche spezifischen Probleme ergeben sich im Requirements Engineering, die in nachgeordneten Arbeitsbereichen nicht auftreten.
2. Betrachten Sie das in im entsprechenden Abschnitt eingeführte alternative SW-Erstellungs-Paradigma “Programmer’s Apprentice” und versuchen Sie, aufgrund der folgenden Fragen, die Schwierigkeiten, die bei der Realisierung eines SW-Systems nach diesem Paradigma entstehen, zu ergründen:
   a) Welche Teile einer Anforderungsdefinition sind überhaupt nicht/zur Zeit nicht formal beschreibbar?
   b) Was kann man über den Detaillierungsgrad einer Anforderungsdefinition im Sinne der in der Vorlesung vorgestellten Figur aussagen?
   c) Halten Sie es für realistisch, dass eine so detailliert beschriebene komplexe Aufgabenstellung, die einem Programm in einer sehr hohen Programmiersprache entspricht, ohne Vorüberlegungen (Anforderungsdefinition bzw. Entwurf der so detailliert festgelegten Aufgabenbeschreibung) erstellt werden kann?
   d) Wie schätzen Sie die Chance ein, dass - selbst wenn man sich einen bestimmten Anwendungsbereich auswählt - das gesamte Wissen über den Entwicklungsprozess aller Programme dieses Bereichs sich in Transformationen gießen läßt und ein System in der Lage ist, automatisch die eigenen Kombinationen der Transformationen herauszufinden?

e) Für welche Aufgabenbereiche und in welchem Zeitrahmen ist eine Vorgehensweise gemäß der oben angesprochenen Figur denkbar?

3. Wenn man eine Lösung eines Problems in einer hohen oder sehr hohen Programmiersprache als Anforderungsdefinition akzeptiert, was z.Z. nur für kleine und überschaubare Probleme geht, dann fügen sich viele der aktuellen Softwareherstellungs-Paradigmen in die Vorstellung des Programmer's Apprentice ein. Solche Paradigmen sind:
   a) Programmieren ist Transformieren, z.B. /Ba 85, 87/,
   b) Programmieren ist Erstellen von Regelsätzen, z.B. /Br 86, Wa 86/,
   c) Programmieren ist Festlegen von Objekten, Botschaften, Klassen und Vererbung, z.B. /Co 86, GR 83/,
   d) Programmieren ist Spezifizieren, z.B. /DGL 79/.

   Wie fügen sich diese Vorstellungen ein? Welche Aufgaben übernimmt der Compiler/Interpreter und welche der Softwareentwickler?
4. Für eingebettete Systeme, bei denen die Software ein wohldefinierter Teil eines technischen Gesamtsystems ist und bei denen die Frage, was in Hardware und was in Software gelöst wird, oft erst im Laufe des Entwicklungsprozesses geklärt wird und später im Verlauf der Wartung abgewandelt wird, ist der SW-Lebenszyklus in einen übergeordneten Systementwicklungszyklus eingebettet. Was liegt vor, bevor der SW-Lebenszyklus beginnt? Wodurch unterscheiden sich solche SW-Prozesse von denen abgeschlossener Systeme (betriebswirtschaftliche Anwendungen, z.B. Buchungssystem)?
5. Charakterisieren Sie eingebettete Systeme und bestimmen Sie damit die Unterschiede zu sonstigen (sequentiellen) Softwaresystemen.

6. Charakterisieren Sie einen Wiederverwendungsprozess eines Systems, das auf eine wohldefinierte Basisplattform aufsetzt.
7. Stellen Sie den Prozess der Neuentwicklung eines Mehrphasencompilers durch ein Prozessnetz dar, so dass alle Teilsysteme durch einen entsprechenden Entwicklungsprozess repräsentiert sind.

Literatur zu Kap. 2:

zur Diskussion von Lebenszyklusmodellen vgl. ST-Einführungen, Angaben in Kap. 0

/Agr 86/ W. Agresti (Ed.): New Paradigms for Software Development, IEEE Comp. Soc. Press, 1986

/Bal 85/ R. Balzer: A 15 Year Perspective on Automatic Programming, Trans. on Software Eng. 11, 11, 1257-1268, 1985

/BCG 83/ R. Balzer, T. Cheatham, C. Green: Software Technologies in the 90's: Using a new paradigm, Computer 11, 39-45, 1983

/BD 93/ A.-P. Bröhl, W.Dröschel (Ed.): Das V-Modell, Oldenbourg Verlag, 1993

/Boe 84/ B.W. Boehm: Software Lifecycle Factors, in Vick/Ramamoorphy (Eds.): Handbook of Software Engineering, 494-518, Van Nostrand Reinhold, 1984

/Boe 86/ B.W. Boehm: A Spiral Model of Software Development and Enhancement, ACM Software Eng. Notes 11, 4, 22-42, 1986

/Flo 89/ Ch. Floyd: Software-Entwicklung als Realitätskonstruktion, Informatik-Fachberichte 212, 1-20, Springer, 1989

/KS 82/ R. Kling, W. Scacci: The Web of Computing: Computing Technology as Social Organization, in Yovits (Ed.): Advances in Computers 21, 1-90, Academic Press, 1982

/MJ 82/ D.D. McCracken, M.A. Jackson: Life Cycle Concepts Considered Harmful, ACM Software Eng. Notes 7, 2, 29-32, 1982

/Nag 90/ M. Nagl: Softwaretechnik: Methodisches Programmieren im Großen, Springer, 1990, Kap. 1 und 2

/Nag 96/ M. Nagl (Ed.): Building Tightly-Integrated Software Development Environments: The IPSEN Approach, LNCS 1170, Springer, 1996

/Nag 99/ M. Nagl: Die Softwaretechnik-Programmiersprache Ada 95, Vieweg, 1999

/Ram 84/ C.V. Ramamoorphy: Software Engineering - Problems and Perspectives, Computer 10, 1984

/RW 90/ C. Ritch, R.C. Waters: The Programmer's Apprentice, ACM Press, 1990

/STA 83/ The DoD STARS Program, Software Technology for Adaptable, Reliable Systems, Sonderheft IEEE Computer, Nov. 1983