Technologie der Software-Entwicklung Stand und Tendenzen

abc Information GmbH

1 .6Die Phasen im Software-Lebenszyklus

Der Software-Lebenszyklus beschreibt als Modell, welche Schritte im Prozeß der Entstehung und Nutzung von Software durchlaufen werden, wobei in der Regel bestimmte Schrittfolgen zyklisch durchlaufen werden. Für dieses Modell gibt es die verschiedensten Darstellungsformen, sowohl was die Wortwahl als auch was die Form der grafischen Darstellung anbetrifft.

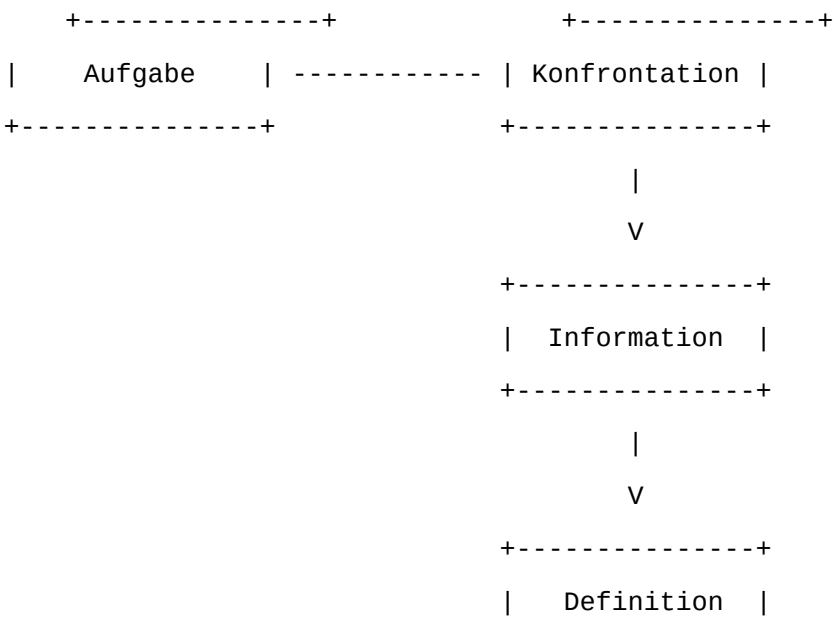
1.1 6.1. Software-Lebenszyklus

Mögliche Blickwinkel

- Software als integraler Bestandteil einer Anwendungslösung
- Software als spezifisches Produkt

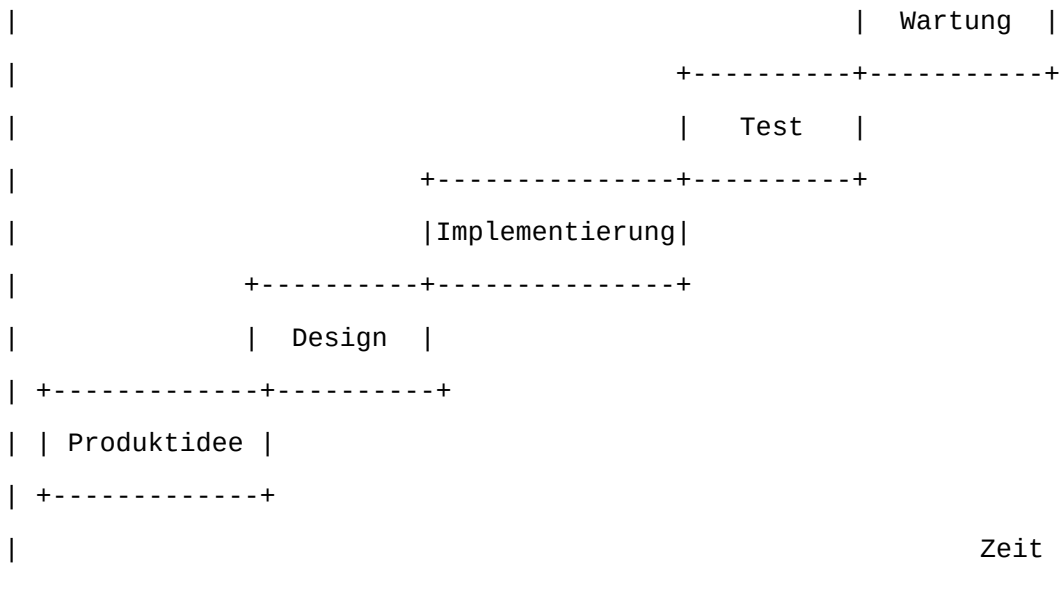
Lebenszyklus von Anwendungssystemen [Lehner 1990]:

- Systementwicklung
- Systemeinführung
- Wachstum
- Reife
- Rückgang



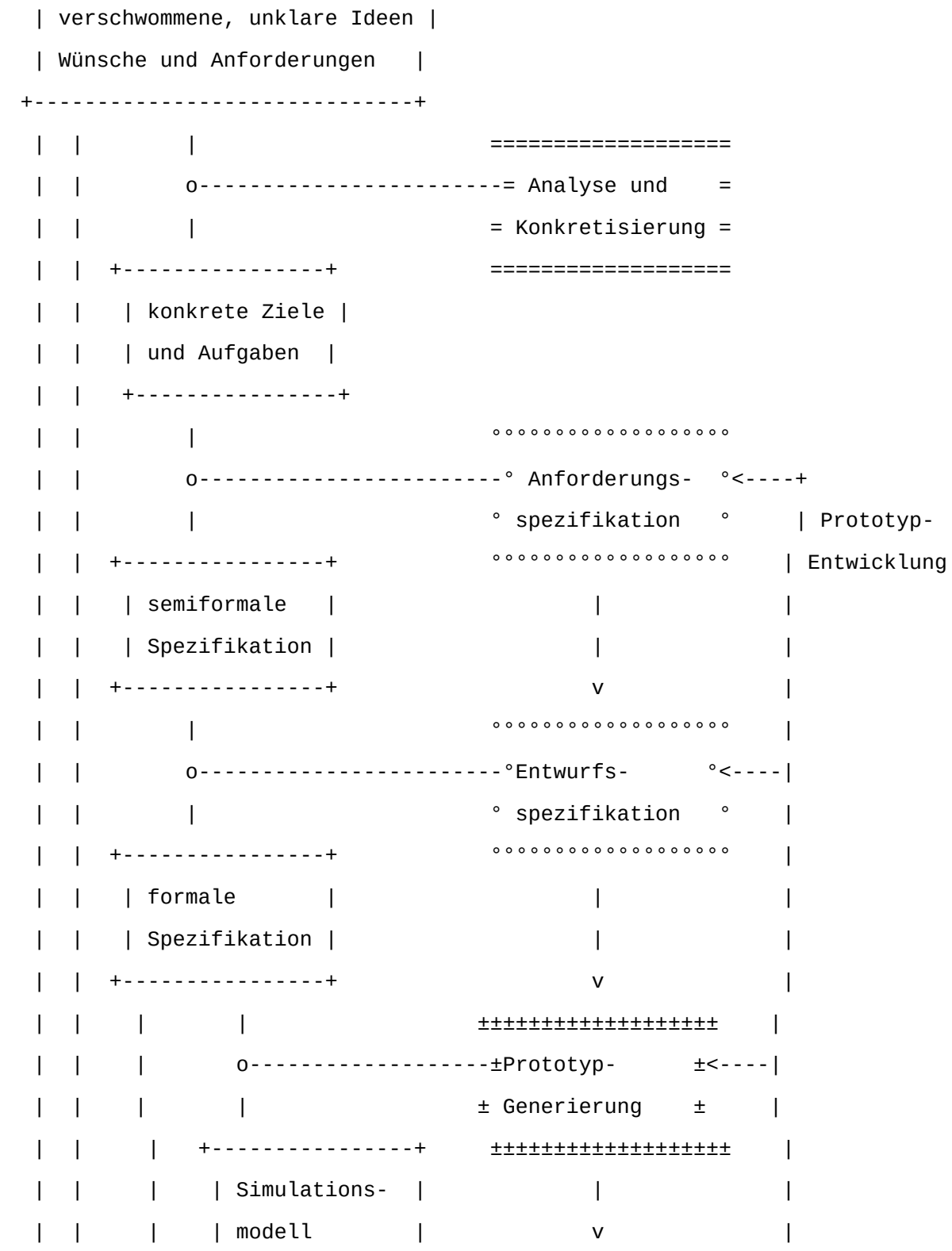

Kreation
Kontrolle,
Beurteilung
Entscheidung
Lösung
Istzustand
Modellierung
deskriptives Modell
(Abbild)
riskante
Modifikation
Modifikation
geplanter Zustand
Anwendung
(Realisierung)
präskriptives Modell
(Vorbild)
Kosten
Software-Enwicklung

[Schm 89]



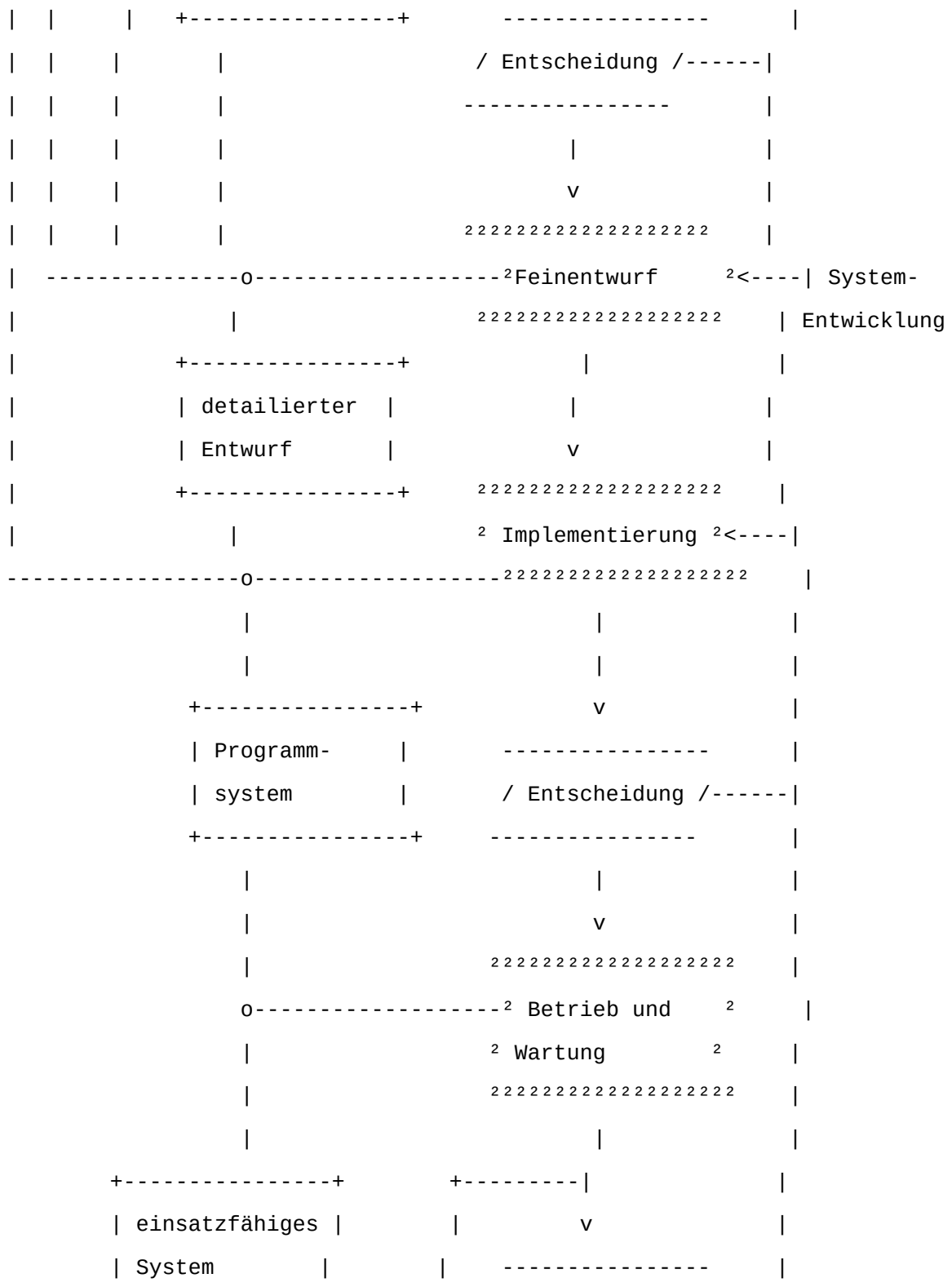
Klassischer Software-Lebenszyklus

Prozesse	Phasen	Teilphasen
Entwicklung	Analysieren	
	Entwerfen	Fachlich-logisches Entwerfen
		Programmtechnisches Entwerfen
	Implementieren	
	Testen	
	Fertigstellen	Bereitstellen
		Erproben
Anwendung	Einführen	
	Betreiben	
	Warten	



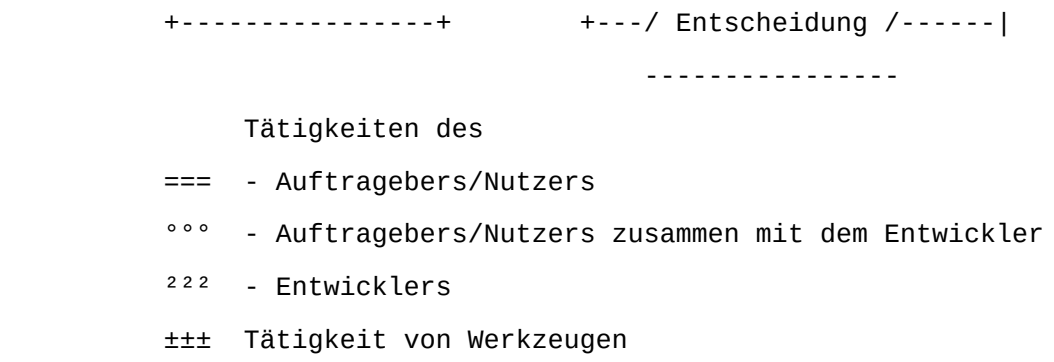
abc Information GmbH
PO Box 130208
13601 Berlin
Germany

+49-700-ITBROKER
http://www.Broker.de
email:contact@Broker.de

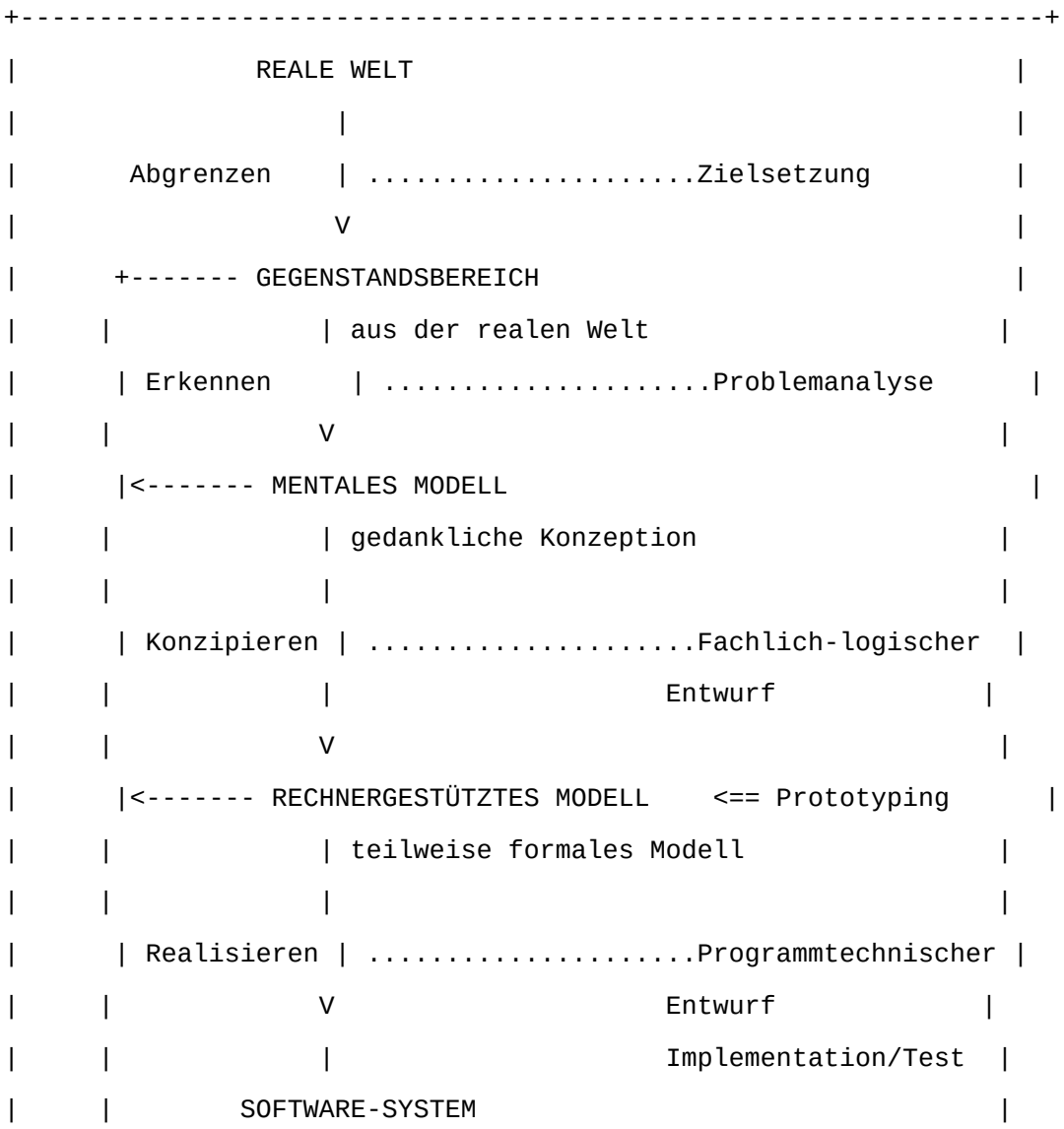


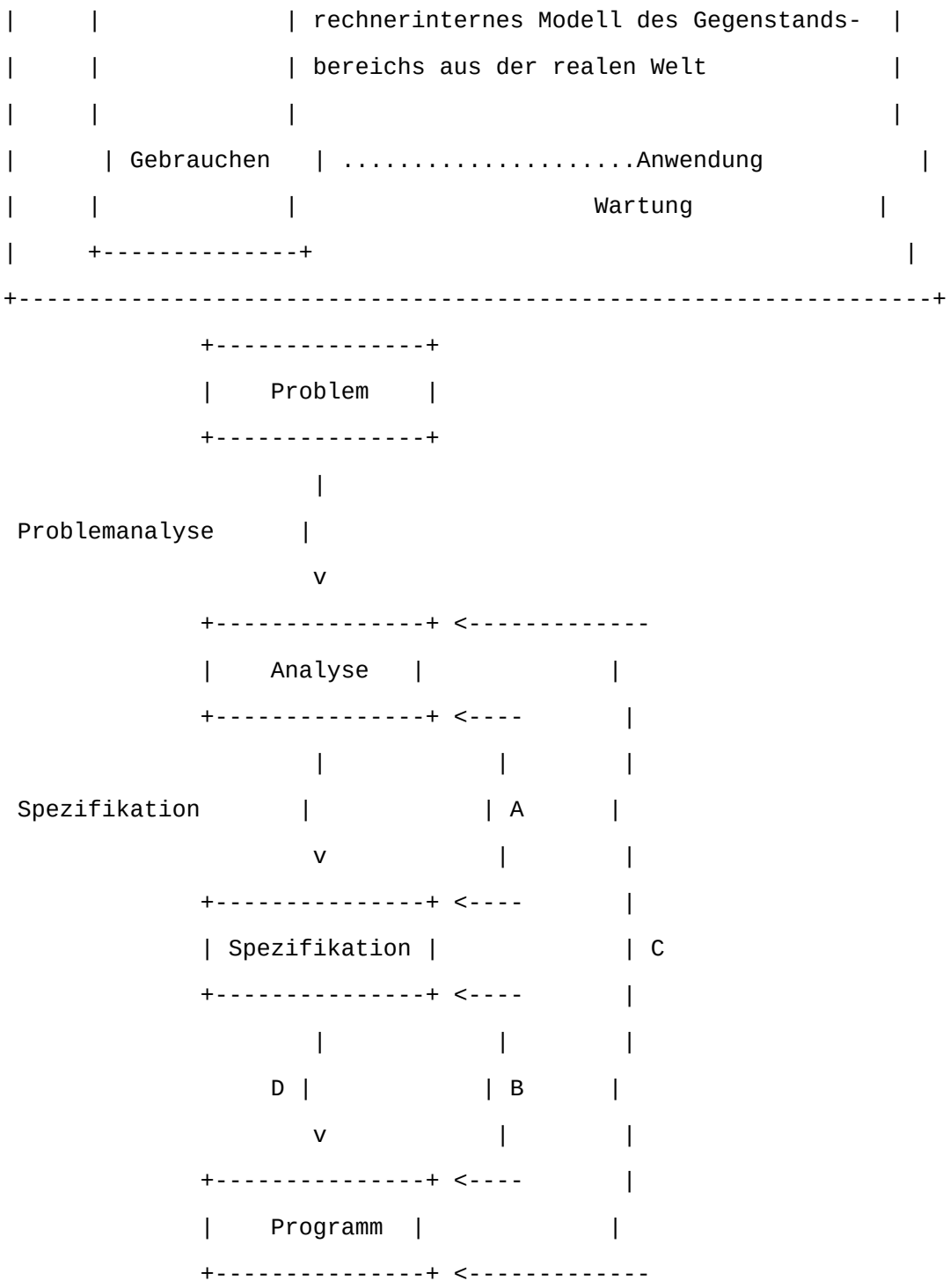
abc Information GmbH
PO Box 130208
13601 Berlin
Germany

+49-700-ITBROKER
http://www.Broker.de
email:contact@Broker.de



Software - Gegenstand und Hilfsmittel der Modellierung
[Böhme 1991]





Es werden 3 Entwicklungsstufen gesehen: 1. Stufe: Die Spezifikation erfolgt informal. Validation, Wartung und Anpassung werden manuell durchgeführt (Zyklen A, B und C). Der Weg von der Spezifikation zum Programm führt über manuellen Entwurf und Implementierung (Pfeil D). 2. Stufe: Die Spezifikation erfolgt formal. Validation, Wartung und Anpassung können jetzt (teilweise) rechnergestützt durchgeführt werden (Zyklen A

und B). Der Zyklus C entf-llt. Der Weg von der Spezifikation zum Programm führt über manuellen Entwurf und Implementierung (Pfeil D). 3. Stufe: Die Spezifikation erfolgt formal. Validation, Wartung und Anpassung werden rechnergestützt durchgeführt (Zyklus A). Auf Grundlage der formalen Spezifikation wird das Programm generiert (Pfeil D), die Zyklen B und C entfallen damit. Die Hineinverlagerung von Testprozessen in die frühen Phasen der Software- Entwicklung werden auch als Prototyping bezeichnet. Die Idee des Prototyping hat vor allem seit Mitte der achtziger Jahre zunehmend an Popularität gewonnen.

Überblick über Analyse- und Beschreibungsmethodik: [BeJW 88]

1.2 6.2.

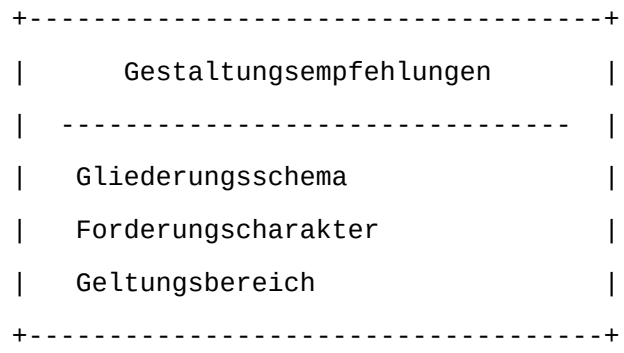
1.3 Analyse

[DeMarco 1978]
Analysis is the study of a problem, prior to taking some action.

Untersuchung - "was" ein System tun muß -- Anforderungen (requirements) - nicht, wie es implementiert werden soll Hauptprobleme bei Systemanalyse [CoYo 90]

- problem space understanding
- person-to-person communication
- continual change

[Encyclopedia Britannica]
In apprehending the real world, men [people] constantly employ three methods of organization, which pervade all their thinking: (1) the differentiation of experience into particuler objects and their attributes - e.g., when they distinguish between a tree and its size or spatial relations to other objects, (2) the distinction between whole objects and their component parts - e.g., when they contrast a tree with its component branches, and (3) the transformation of and the distinction between different classes - e.g., when they form the class of all trees and the class of all stones and distinguish between them. Systemanalyse - Einige Aspekte Aspekt 1 - oft Tendenz zur Betrachtung als rein technisches System - notwendig Betrachtung als soziotechnisches System, d.h. als soziales und technisches System Aspekt 2 - oft Tendenz zur ausschlie+lichen bzw. vorwiegenden Analyse des Ist-Zustandes mit dem Ziel, vorhandene Strukturen auf einem höherem technischen Niveau zu erhalten - mögliche Wirkung: Schwachstellen werden effektiver wirksam - notwendig Schwachstellenanalyse Bereitschaft, neue Strukturen zu entwickeln Problem: Umgang mit dem oft vernünftigen Grundsatz never change a running system Frage: Was ist riskanter, etwas zu verändern, oder nichts zu verändern ? Modell darf nicht nur aus 'technischen' Objekten bestehen, es mu+ in geeigneter Weise auch die 'menschlichen' Objekte berücksichtigen Grundsatz: Je komplizierter und komplexer ein technisches System, desto wichtiger ist es, seine Wirkung auf den Menschen zu untersuchen [Stei 93]
Die ersten strukturierten Analysemethoden wurden vor 15 Jahren publiziert und anschließend konsequent weiterentwickelt. Die letzte signifikante Erweiterung wurde 1987 von Hatley und Pirbhai vorgenommen. Heute ist das Gebiet der strukturierten Methoden konsilidiert.
In England und Frankreich haben sich die Methoden SSADM bzw. Merise etabliert. In Deutschland gibt es z.Zt. keinen offiziellen Standard, aber SA nach DeMarco [DeMa 78] ist als de-facto-Standard akzeptiert.



1.4 6.3. Entwurf

[Eber 92] ... nur vier Grundparadigmen für die Entwurfsbeschreibung erforderlich sind, von denen die existierenden Entwurfsbeschreibungssprachen konkrete Varianten sind.

Entity-Relationship-Beschreibungen

> Hier werden Realitätsausschnitte durch die Beschreibungen von bedeutungstragenden "Informationsobjekten" und deren Beziehungen dargestellt.

Datenfluß-Beschreibungen

> Hier wird beschrieben, wie mit Hilfe von Prozessen Informationen durch ein Geflecht von Informationsspeichern bewegt werden.

Kontrollfluß-Beschreibungen

> Hier wird die Durchführung komplexer Aktionen durch sequentielle (evtl. auch alternative oder wiederholte) Ausführung anderer einfacher Aktionen dargestellt.

Zustands-Übergangs-Beschreibungen

> Hier wird die Abhängigkeit der Reaktion von Systemen auf äußere Ereignisse von deren internem Zustand dargestellt.

1.5 6.4. Prototyping

Während der klassische Software-Lebenszyklus ohne Prototyping auskam, hat sich die Idee des Prototyping inzwischen recht weit verbreitet.
Nur leider [Hall 90]
... wird der Begriff 'Prototyping' in der Informatik sehr unterschiedlich diskutiert.

Das Prototyping-Modell geht dabei, im Unterschied zum klassischen Modell des Software- Lebenszyklus, von folgenden Annahmen aus:

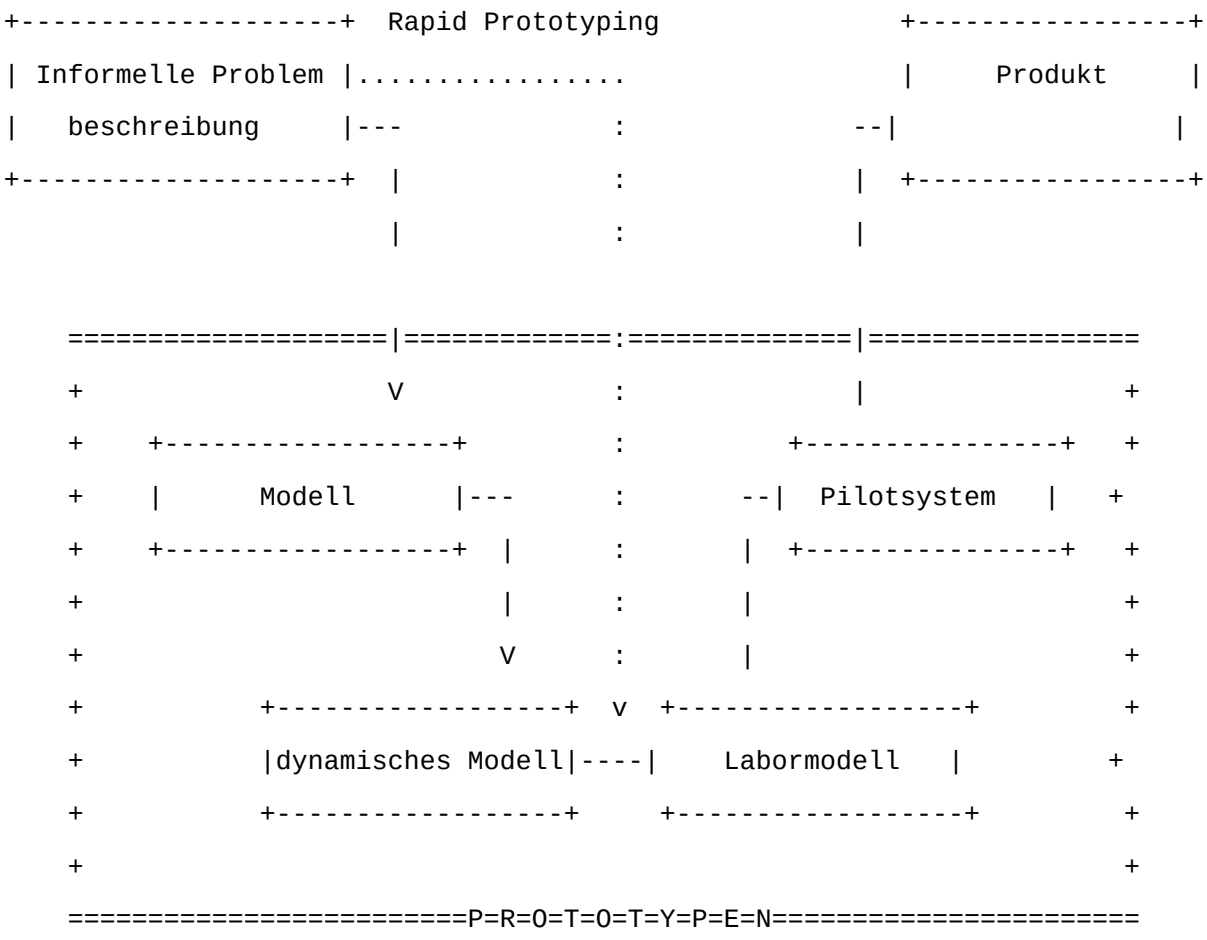
- die wahren Anforderungen des Anwenders lassen sich von vorn herein nicht vollständig feststellen,
- sowohl die Organisation in welche die Software eingebettet ist, als auch die Software selbst werden nicht als unveränderlich (statisch) angesehen, sondern als dynamisch,
- die Software wirkt auf den Anwender ein und bestimmt somit dessen Sicht auf sein Arbeitsumfeld, d.h. auch dessen Anforderungen an die Software selbst (Regelkreiseffekt).

Der Begriff Prototyp wurde ursprünglich in den Ingenieurwissenschaften geprägt [Hall 90]:
Dort wird mit Prototyp das Produkt bezeichnet, das zwischen Zeichenbrett und Massenfertigung erstellt wird und alle Merkmale des späteren Verkaufsprodukts enthält. Das Produkt Software und deren Serienproduktion unterscheiden sich allerdings in wesentlichen Merkmalen ...

[Snee 85]
In der Literatur wird oft von "Prototyping" gesprochen. Das Ergebnis der ersten Stufe könnte als Prototyp bezeichnet werden. Aber hier hinkt der Vergleich zu Prototypen in anderen Ingenieurdisziplinen. Dort ist das Ergebnis ein verkleinertes Modell des endgültigen Produktes zum Zwecke der Planung. Im Falle von Software-Systemen ist das nicht zutreffend. Die erste Version ist ein unvollkommenes Produkt, mit dem man schon arbeiten kann. Sie hat aber nicht die Eigenschaften, die man sich langfristig wünscht.

Deshalb ist auch die folgende Anmerkung nicht ganz unberechtigt [Lude 89]:
"Prototyp" ist für Software ein unglückliches Wort...

Einer der wesentlichsten Unterschiede liegt in folgendem [Resi 89]:
In gewisser Weise ist Softwareentwicklung als "Orginalentwicklung" zu charakterisieren. Wir sagen ja bewußt nicht "Software-Fertigung", sondern "Software-Entwicklungs-Projekt".



Während der Prototyp der 'klassischen' Industrie mit dem Endprodukt (im wesentlichen) identisch ist, ist der Prototyp der Software-Industrie ein Modell. Das folgende, auf Kreplin zurückgehende Schema versucht eine Klassifikation des Prototyping [Vorw 91]:

Zweck des Prototypen		
erforschend	experimentierend	evolutionär
analysierend	evaluierend	
Dient als Mittel zur Anforderungsanalyse und Aufstellen des Pflichtenheftes	Wird zur Prüfung benutzerkonzipierter Lösungen benutzt und kann als Verfeinerung der Anforderungsdefinition gesehen werden	Softwareentwicklung, die auf verschiedenen Versionen basiert, mit denen der Anwender arbeiten kann

Weiterverwendbarkeit des Prototypen

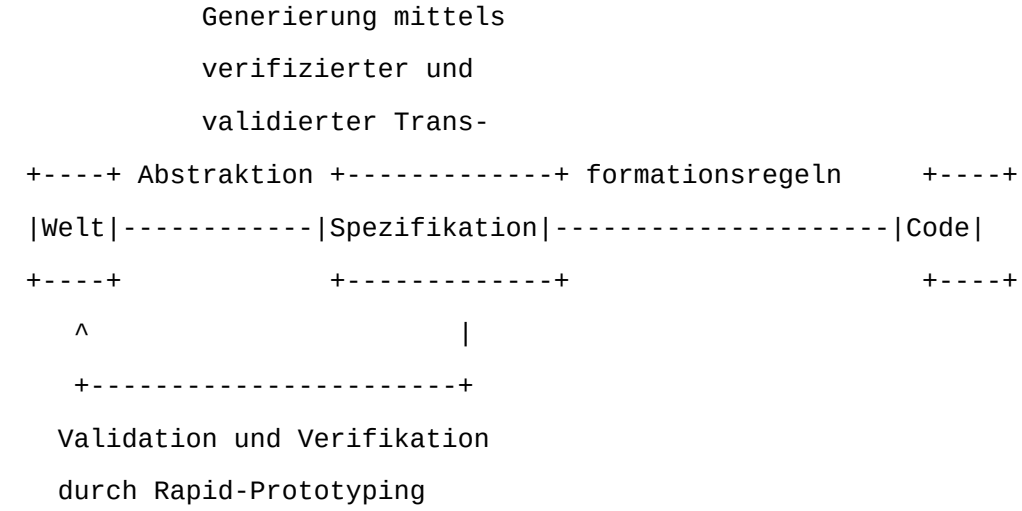
"Wegwerf"-Prototyp	Prototyping für Benutzertraining	Basis für Systementwicklungen
Schnelle Realisierung einer Oberfläche	An der Anwenderoberfläche ist ausreichender Funktionsumfang vorhanden	Weiterentwicklung zur Nutzung des bereits geleisteten Implementierungsaufwandes

Ziel des Prototypen

Unterstützung der Benutzerschnittstelle	Unterstützung des Systems
Prototyping der Anwenderschnittstelle als wichtiger Ansatzpunkt der Software-Ergonomie	Prototyping zum Festlegen, Implementieren und Überprüfen von Systemeigenschaften

Eine Kurzcharakteristik verschiedener Formen von Prototyping findet sich in [Lude 89]. Wunschvorstellung ist es, aus der Software-Spezifikation heraus zu einem ablauff-higen Prototypen zu kommen. Dies setzt eine formale Spezifikation voraus und wird als Rapid Prototyping im engeren Sinne bezeichnet.

Prototyping ist in verschiedenen Phasen und Formen möglich. So wird beispielsweise in [ToHa 86] unterschieden in Prototyping - in der Definitionsphase, - in der Entwurfsphase und - in der Implementierungsphase. In der Definitionsphase geht es dabei vor allem um ein Testen der Nutzerschnittstelle. Nach Floyd [Floy 89] kann Prototyping folgende Formen annehmen: - explorativ, - experimentell, - evolutionär. Die verschiedenen Formen sind nicht als Gegens-tze aufzufassen, sondern sie können sich erg-nzen bzw. ablösen. [GmSa 90]



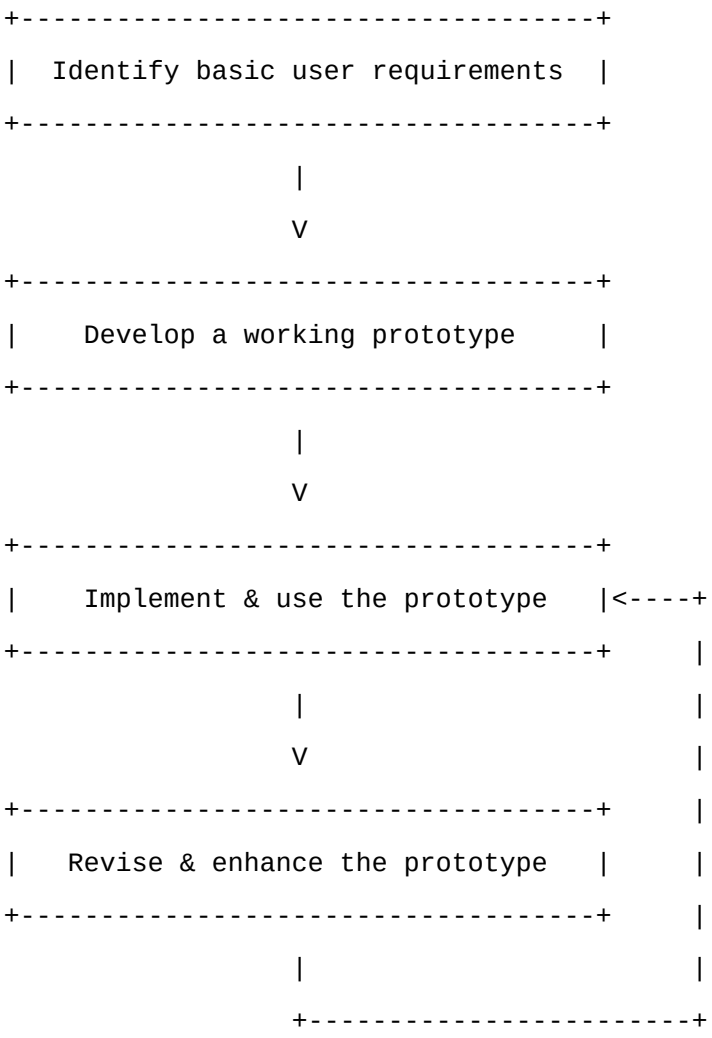
Lantz [Lant 86] versucht, aus den Problemen, die mit der klassischen Vorgehensweise verbunden sind, eine Motivation für Prototyping abzuleiten: Problems with current methodologies

- Are throrough	BUT DON'T	pleases users
- Produce extensive documentation	BUT DON'T	decrease communication problems
- Identify project steps	BUT DON'T	decrease calendar time needed
- Describe system thorougly	BUT DON'T	guarantee it's the right system
- Delineate skills needed	BUT DON'T	cut manpower needed
- Track project costs	BUT DON'T	reduce them

Zu beachten ist natürlich, da+ der Einsatz von Prototyping allein die genannten Probleme nicht löst. Differenziert gesehen werden sollte die folgende Auffassung von Lantz [Lant 86]: The initial model of a prototype should feel as much as possible like the final version of the system Prototyping sollte immer in Zusammenarbeit mit den

künftigen Nutzern erfolgen. Dabei hängt es wesentlich von dem betreffenden Partner ab, ab welchem Reifegrad des Modells er in den Prototyping-Proze+ einbezogen werden sollte. Ein zu frühes Einbeziehen kann dem Ansehen des Entwicklers schaden, ein zu spätes Einbeziehen den Entwicklungsproze+ verlängern. Prototyping muß nicht erst beginnen, wenn das Gesamtmodell vorliegt. Wenn möglich sollte bereits auf Modellebene modularisiert werden, um frühestmöglich, gegebenenfalls auch parallel, mit dem Prototyping von Teilmodellen beginnen zu können.

[SrRa 86]
When the users determine that no further changes need to be made, developers must decide what do do with the prototype. Three alternatives exist: - Use the prototype as a design specification for development of the production system. - Put the prototype into production and use it as the actual system. - Scrap the prototype. Four phase prototyping procedures



[Hopp 88]
Das aktuelle Schlagwort Rapid Prototyping (schnelle Prototypentwicklung) bezeichnet eine Arbeitsweise, die

insbesondere bei der Entwicklung von interaktiven Benutzerschnittstellen wie auch Expertensystemen erfolgreich angewandt wird. Dabei wird im Gegensatz zu einem top-down-Ansatz nicht von einem vollst-ndigen Systementwurf ausgegangen, der zunehmend verfeinert und schlie+lich in complierbaren Code überführt wird. Vielmehr ist die schnelle Prototypentwicklung durch eine inkrementelle Vorgehensweise gekennzeichnet: Ausgehend von einer konzeptionellen Vorstellung davon, was das geplante System leisten soll, werden wesentliche Funktionskomponenten herausgegriffen und unter weitgehender Vermeidung von Details in Form lauff-higer Module programmtechnisch realisiert. Diese Bausteine werden im interaktiven Betrieb getestet, verbessert, erweitert und zu größeren Einheiten zusammengefügt.

1.6 3.6.2. Objektorientierte Analyse und objektorientierter Entwurf

Während bei der objektorientierten Programmierung bereits 'der Stein ins Rollen' gekommen ist, sieht es in den vorgelagerten Phasen mit der Objektorientiertheit noch bescheidener aus. Keramidis [Kera 91] hat in einer Diskussion zur CASE-Entwicklung folgende Einsch-tzung gegeben:

		damals	1998	2002
Objektorientierung	--	-	+	

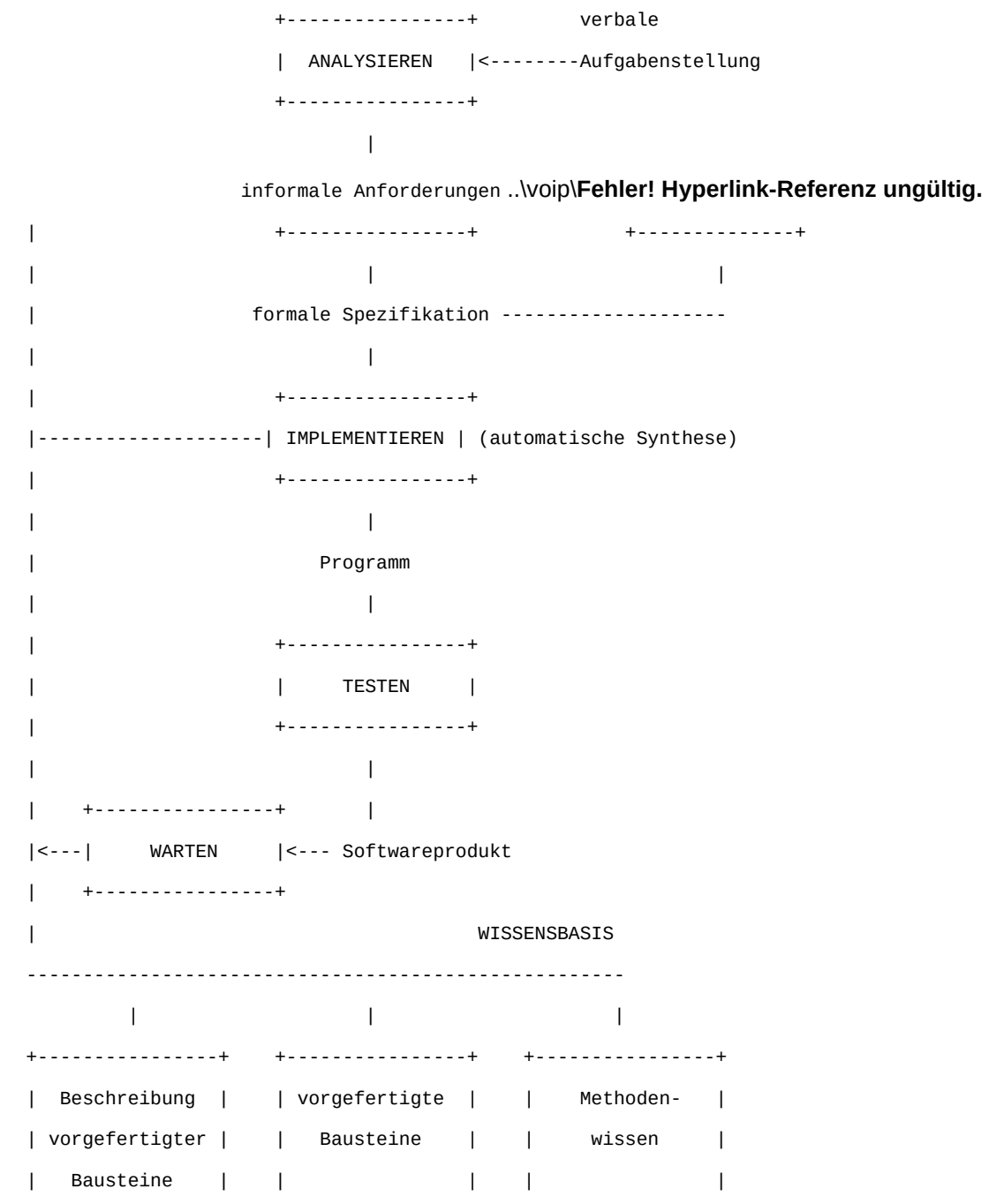
Dies liegt zum einem daran, daß die objektorientierte Idee für Analyse und Entwurf erst relativ spät aufgegriffen wurde, und zum anderen daran, daß sich Objektorientierung in der großen Mehrzahl der verfügbaren Publikationen fast ausschließlich als objektorientiertes Programmieren präsentiert. Sowohl in der Literatur als vor allem auch in der Praxis der Software-Entwicklung spielen noch immer die Methoden der strukturierten Analyse und des strukturierten Entwurfs die entscheidende Rolle. Aber immerhin ist auch bei den objektorientierten Methoden schon einige Arbeit geleistet worden, siehe z.B. [CoYo 90], [CoYo 91], [ShMe 89]. Stoyan [Stoy 89] kennzeichnet objektorientierte Systementwicklung wie folgt: Objektorientierte Systementwicklung bedeutet Modellierung der Anwendungs welt: Die real-physikalischen Objekte - aber auch abstrakt konstruierten - dieser Welt werden durch Software-Objekte modelliert. [Stoy 89] Die folgenden Schritte sind beim objektorientierten Entwurf vorzunehmen: 1. Bestimmung der Objektklassen 2. Bestimmung der Grundoperationen (Konstruktion, Selektion, Modifikation, Attribution, Vergleich, Konversion) für jede Objektklasse 3. Bestimmung der Systemfunktionen durch Zuweisung von Operationen an eine Objektklasse 4. Implementation der erarbeiteten Operationen durch erneuten objektorientierten Entwurf eine Ebene tiefer 5. Abstieg bis zur Maschinen-/Programmiersprachebene Objektorientiertes Arbeiten erfordert, wenn es erfolgreich sein soll, einen gewissen Bruch mit bisherigen Vorgehensweisen. Dazu Taylor [Tayl 91b]:
... the question was "What is the worst thing you can do if you want to get the full benefits of object technology ?" ... the correct answer was "Apply the standard development lifecycle"

Objektorientiertes Arbeiten nimmt aber auch bisherige Arbeitsweisen in sich auf. Auch hierzu Taylor [Tayl 91b]: Note, however, that I did not advocate throwing out all the traditional stages. You can't build quality software without doing solid requirements analysis and good system design, nor can you blithely ignore the minor nuisances of testing and maintenance. The trick is to keep all the traditional stages but to apply them individually to every class, model, and application you build. This technique allows you to reuse not just code but all the analysis, design, testing, and maintenance that went into the lower levels of your system.

1.7 3.6.3. Formale Spezifikation

Mit der formalen Spezifikation werden (perspektivisch) vor allem zwei Zielvorstellungen verbunden: - Erbringen des Beweises, daß eine formulierte Spezifikation bestimmten Anforderungen tatsächlich gerecht wird. - Nutzung der formalen Spezifikation als Ausgangspunkt für die Generierung einer lauffähigen Software-Version. Das erste Ziel (Validation) ist gegenwärtig nur in sehr kleinen Teilbereichen erreichbar, und dies auch

nur mit ganz erheblichem Aufwand. Beim zweiten Ziel stehen die Chanchen etwas besser: Für bestimmte Aufgaben ist durchaus die Erzeugung von lauff-higem Code aus einer Spezifikation heraus möglich. Dieses Vorgehen wird auch als Rapid Prototyping bezeichnet. Erzeugt wird ein Software-Prototyp, der meist nur einige wesentliche, aber nicht alle geforderten Eigenschaften des Endprodukts aufweist. Experimente mit dem Prototyp ermöglichen es - im Zusammenwirken mit den künftigen Nutzern des Software-Produkts - frühzeitig Spezifikationsfehler aufzudecken. Forbrig [Forb 90] schlägt den folgenden, in der Zukunft anzustrebenden, Lebenszyklus vor:



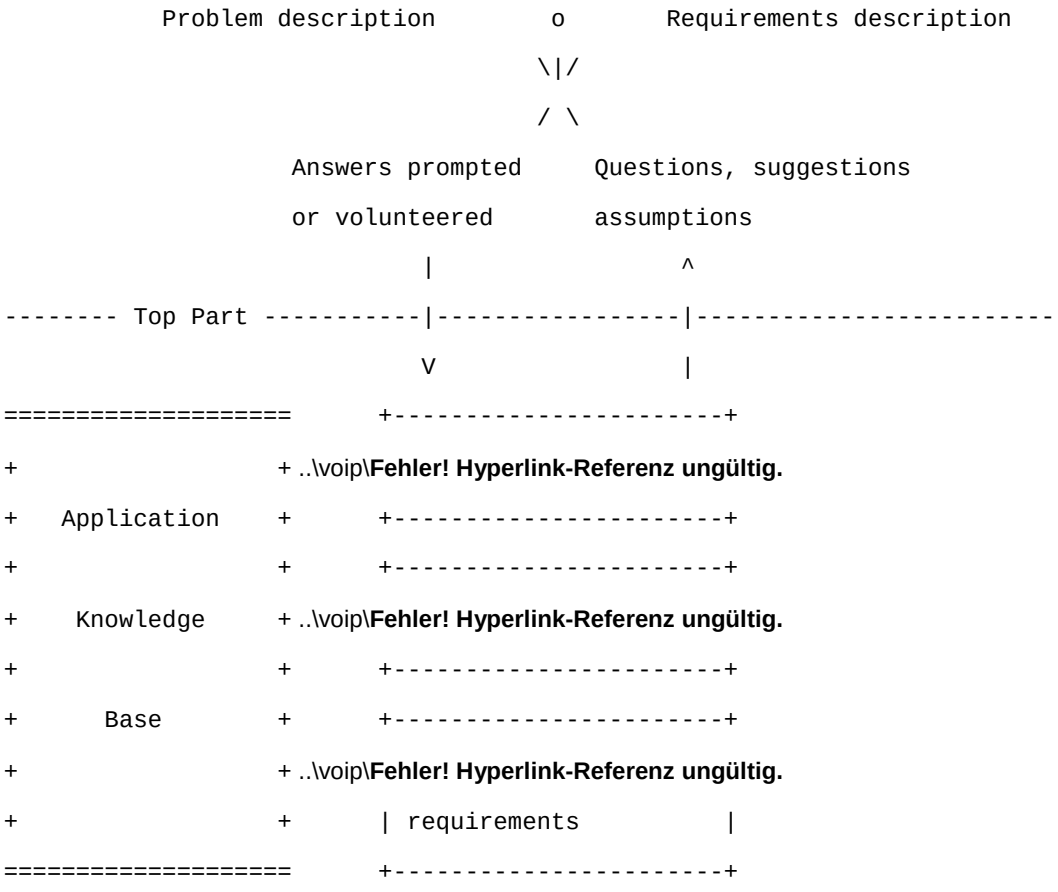
Der Ansatz setzt auf die Verwendung vorgefertigter Software-Bausteine, die entsprechend der speziellen Anforderungen wissensgestützt ausgewählt und verknüpft werden. Voraus- setzung ist natürlich, das nicht nur die Software-Bausteine vorhanden sind, sondern auch die zu ihnen gehörende formale Spezifikation.

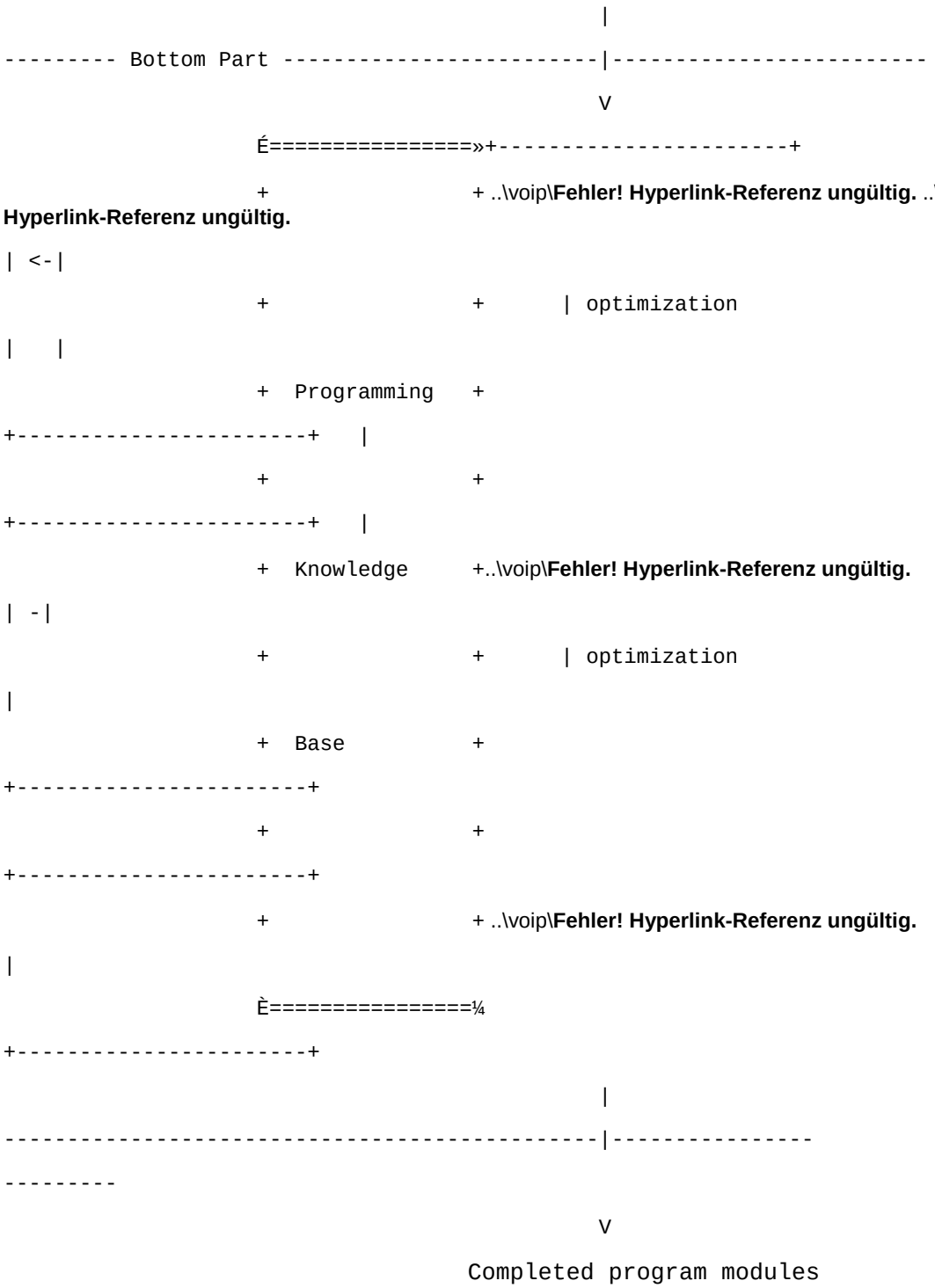
[HoPi 89]
Die formale Spezifikation größerer Softwareprodukte wird sich wahrscheinlich auf Basissoftware und Software, die hohe Sicherheitsanforderungen erfüllen muß, konzentrieren. Dazu wird sich im Proze+ der Arbeitsteilung der Spezialist für formale Spezifikation, der in den entsprechenden theoretischen Grundlagen ausgebildet und in den Spezifikationstechniken geschult werden mu+, herausbilden. Bei der Spezifikation von Anwendersoftware wird erwartet, daß wissensbasierte Methoden und Rapid Prototyping in die ingenieurmäßige Praxis Einzug halten.

[BrLM 85]
Bei der Programmverifikation sind zwei Zielrichtungen zu unterscheiden: 1) Prüfung, ob das Verhalten des Programms mit seiner Spezifikation vertr-glich ist, 2) Prüfung, ob das Programm endet. Die Durchführung des Beweises der Verifikationsbedingungen für ein Programm erfordert in der Regel grö+ere Anstrengungen als die Programmierung selbst.

1.8 3.6.5. Automatische Programmsynthese

Überlegungen zur automatischen Programmsynthese reichen bereits relativ lange zurück, wie zum Beispiel das folgende Schema von Prywes [Pryw 77] zeigt:





Die Generierung von Programmen hat jedoch auch heute noch einen relativ geringen Anteil an der neu entstehenden Software. Hier werden jedoch in nächster Zeit durchaus Änderungen erwartet.

1.9 Wartung

[Snee 85]
Im Gegensatz zu ... ist das Gebiet der Software-Wartung gleich einem Urwald. Es fehlen nicht nur die Methoden und theoretischen Grundsätze, es fehlt auch an Definitionen, was Wartung überhaupt ist.

Swanson definiert vier Funktionen der Software-Wartung:

- Korrigieren
- Anpassen
- Erweitern
- Optimieren