Martin Engelien/Heinz Stahn **Software-Engineering**

ARS-Technologie

Software-Engineering

ARS-Technologie

Von
MARTIN ENGELIEN
und
HEINZ STAHN

Mit 120 Abbildungen

AKADEMIE-VERLAG · BERLIN
1984

Verfasser:

Doz. Dr. sc. techn. Martin Engelien
Prof. Dr. sc. techn. Heinz Stahn
Technische Universität Dresden

Der Titel wurde vom Originalmanuskript der Autoren reproduziert.

Erschienen im Akademie-Verlag, DDR-1086 Berlin, Leipziger Straße 3—4

Lizenznummer: 202 · 100/411/84
Printed in the German Democratic Republic
Gestaltung: Karl Salzbrunn
Fotomechanischer Nachdruck: VEB Druckerei „Thomas Müntzer", 5820 Bad Langensalza
Lektor: Dipl.-Phys. Gisela Lagowitz
LSV: 1084
Bestellnummer: 763 231 2 (6767)

03600

Vorwort

Den aktuellen Anforderungen an Qualität und Quantität von Software für die automatisierte Informationsverarbeitung und Steuerung können wir mit den konventionellen Vorgehensweisen allein nicht mehr gerecht werden. Eine moderne Technologie der Softwareproduktion soll theoretisch begründet und ingenieurmäßig praktikabel sein. Insbesondere gilt es, die der Ausführung vorgelagerte Planung zu qualifizieren und einem breiteren Personenkreis den Zugang zur Softwareproduktion zu ermöglichen. Ähnlich wie im Bauwesen oder im Maschinenbau, soll nun auch in der Softwareproduktion das Produkt Programm nach einem Plan ausgeführt werden, welcher aus der geforderten Funktion abgeleitet wird, selbst (Zwischen-)Produktcharakter hat, bewertbar und überprüfbar ist, die Einbeziehung von Auftraggeber und Nutzer gestattet und neben einer Reihe weiterer Eigenschaften mit geringerem Aufwand sicher zu besserer Software führt. Diesen Anforderungen wird man in den letzten Jahren in zunehmendem Maße durch eine datenflußorientierte Vorgehensweise zur Softwareproduktion gerecht, die der überkommenen rein steuerflußorientierten Denkweise vorgeordnet wird. Der Zwang zu ihrer Durchsetzung ergibt sich aus den Anforderungen an Qualität und Produktivität der Softwareproduktion sowie aus der Notwendigkeit verteilter Informationsverarbeitung und Steuerung.

Die Technologie der allgemeinen und rekursiven Strukturierung (ARS-Technologie) ist eine datenflußorientierte Vorgehensweise zur Softwareproduktion. Im vorliegenden Buch werden ihre Notwendigkeit begründet, eine Formalisierung und theoretische Begründung aufgebaut sowie Elemente, Synchronisationsstrukturen und Koordinationsstrukturen als inhaltliche Schwerpunkte dargestellt. Beispiele sowie eine Bewertung von Qualität und Produktivität illustrieren die Darstellung und dokumentieren die Vorzüge der Vorgehensweise. Die Autoren hoffen, mit dieser Darstellung einen breiten Kreis an der Softwareproduktion interessierter Leser anzusprechen und der datenflußorientierten Vorgehensweise im Software-Engineering zu einer umfassenden Durchsetzung zu verhelfen.

Wir danken dem Akademie-Verlag und insbesondere der Lektorin, Frau Dipl.-Phys. G. Lagowitz, für die verständnisvolle Zusammenarbeit.

Dresden, Januar 1984 M. Engelien und H. Stahn

Inhaltsverzeichnis

1. EINFÜHRUNG

1.1. Gegenstand, Zielstellung und Grobcharakteristik

Software ist die konstruktive informationelle, meist programmtechnische Realisierung der Verhaltensfunktion, im weiteren Prozeß genannt, automatisierter Systeme der Informationsverarbeitung oder Steuerung und stellt in Einheit mit stofflich-energetischen Trägern solche Systeme dar.

Software-Engineering ist die Ingenieurwissenschaft von den Gesetzmäßigkeiten, Methoden und Mitteln zur zweckmäßigen Gestaltung und Realisierung von Prozessen zur Produktion von Software.

ARS-Technologie ist eine in der Allgemeinen Kybernetik begründete Vorgehensweise zur Produktion von Software durch Allgemeine und Rekursive Strukturierung der Prozesse der Transformation von Information, d.h. von Prozessen der Informationsverarbeitung oder Steuerung.

Derartige Vorgehensweisen werden heute als funktional, meldungs- oder datenflußorientiert bezeichnet ("data driven execution of purely functionaly operations of a dataflow structure") und der konventionellen steuerflußorientierten Vorgehensweise ("control-flow structuring") gegenübergestellt, die nicht mehr genügt, die heutigen Probleme der Informationsverarbeitung und Steuerung zu lösen. Ihre besondere Bedeutung hat die ARS-Vorgehensweise für die heute durch die Mikroelektronik wirtschaftlich gewordene Möglichkeit zur Realisierung von verteilten Steuerungen und Informationsverarbeitungsprozessen mit Mehrprozessorsystemen.

Wesentliche aktuelle Probleme der Softwareproduktion sind folgende:

- Verbesserung der Produktivität der Softwareproduktion,
- Verbesserung der Qualität der Softwareprodukte.

Mit der weitestgehend auch quantitativ bewertbaren Lösung der beiden Hauptprobleme sind folgende Forderungen zu erfüllen:

- Durch den breiten Einsatz der Mikroelektronik bedingt, sind in stark zunehmendem Maße Personen aus Disziplinen außerhalb des Software-Engineering in die Softwareproduktion einzube-

ziehen, insbesondere in die Produktplanung. Daraus folgt u.a., daß die hierbei eingesetzten Methoden und Mittel auf interdisziplinären Abstraktionen basieren sollten.

- Der Prozeß der Produktion von Software soll planbar und kontrollierbar und seine Zwischenprodukte sollen abhebbar und bewertbar gestaltet werden.
- Bereits bei der Produktion ist zu gewährleisten, daß Software an sich entwickelnde Nutzungsbedingungen und -anforderungen anpaßbar gestaltet wird.
- Die Softwareproduktion selber ist zu automatisieren. Entsprechende Mittel hierfür werden heute als Tools bezeichnet.

Die ARS-Technologie stellt einen signifikanten und nachweisbaren Beitrag zur Lösung der genannten Hauptprobleme und Forderungen dar. Sie zeichnet sich hierbei durch folgendes aus:

- explizite und konsequente Nutzung von Erkenntnissen der Allgemeinen Kybernetik zwecks Ableitung einer theoretisch begründeten Vorgehensweise und interdisziplinärer Abstraktionen mit Allgemeingültigkeit;
- mengentheoretisch-algebraische Formalisierung der datenflußorientierten Vorgehensweise ARS-Technologie unter Einbeziehung bekannter und entwickelter Gesetzmäßigkeiten informationeller Strukturen, um formale Strukturtransformationen und Konsistenzprüfungen sowie die Erstellung von Tools zu ermöglichen;
- quantitative Bewertung von Produktivität und Qualität für Schritte, Zwischenergebnisse und Technologien/Vorgehensweisen der Softwareproduktion zum Zwecke des Vergleiches, der Kontrolle und einer zielgerichteten Beeinflussung;
- widerspruchsfreie Integration der konventionellen steuerflußorientierten Software-Techniken als notwendige spezielle Vorgehensweisen.

Die Anwendung der ARS-Technolgie und ihr quantitativ bewerteter Erfolg werden anhand einer Reihe typischer Beispiele demonstriert. Die Anwendbarkeit der ARS-Technologie und ihre Vorzüge erstrecken sich von der Projektierung (Requirement Analysis) über die Konstruktion (Programm Synthesis) bis hin zur Nutzung und Änderung von Software.

Der Schwerpunkt der Darstellung im vorliegenden Buch liegt, wie im weiteren begründet wird, auf der Projektierung, womit die der programmtechnischen Ausführung vorgelagerte Planung eines Softwareproduktes gemeint ist. Die Mehrzahl der Beispiele ist bis zur Implementierung und Nutzung ausgeführt.
Die Darstellung der ARS-Technologie in der interdisziplinären mengentheoretisch-algebraischen Sprache der Allgemeinen Kybernetik bedeutet nicht, daß für ihr Verständnis keine Voraussetzungen notwendig sind. Voraussetzungen sind etwa im Umfange des Kurses "Digitale Systeme - Grundlagen" nach WUNSCH/SCHREIBER, wie er in der Ausbildung einiger ingenieurwissenschaftlicher Fachrichtungen geboten wird, wünschenswert. Um jedoch auch den Leser mit einer in geringerem Maße auf der Diskreten Mathematik aufbauenden Ausbildung anzusprechen, wurde versucht, einen Teil (insbesondere Kap. 2.) weitestgehend ohne mengentheoretisch-algebraische Formalismen darzustellen. Des weiteren soll die Vielzahl der graphischen Darstellungen zu den Beispielen einen Zugang zur ARS-Technologie erleichtern. Hierbei geht es uns sowohl bei der allgemeinen Darstellung der ARS-Technologie wie auch in den Beispielen primär um die Darstellung der Gesetzmäßigkeiten und Methoden sowie ihre Demonstration in Prinziplösungen der Beispiele, weil deren Verständnis entscheidend für die effektive Anwendung der ARS-Technologie ist. Projektierungsvorschriften und Standards wären auf dieser Basis abzuleiten und werden wesentlich von den betrieblichen Einsatzbedingungen bestimmt.
Ein weiterer, aus der bisherigen Vermittlung der ARS-Technologie und der internationalen Literatur bekannter Umstand ist, daß die neue Qualität einer datenflußorientierten Denkweise im Vergleich zur konventionellen steuerflußorientierten ein bestimmtes Maß an Bereitschaft zum Umdenken erfordert. Das gilt insbesondere für Leser mit einer ausgeprägten steuerflußorientierten Programmierungspraxis (vgl. BACKUS: "Befreiung vom Von-NEUMANN-Stile" oder KIMM/KOCH/SIMONSMEIER/TONTSCH: "Methode fällt aus dem Rahmen der bisher betrachteten", Problem "Bit-Fummler ablösen").
Die ARS-Technologie in der vorliegenden Darstellung ist aus der Arbeit der Autoren zu Entwurf, Implementierung und Überführung automatisierter Systeme der Informationsverarbeitung und Prozeßsteuerung sowie einer entsprechenden Lehrtätigkeit erwachsen. Hierbei wurde für Steuerungen diskreter Prozesse vorzugsweise

die Entscheidungstabellentechnik eingesetzt. Dieser Umstand prägt auch die vorliegende Darstellung, insbesondere die konstruktive Darstellung in den Beispielen. Die gute Eignung der Entscheidungstabellentechnik für den gesamten Prozeß der Softwareproduktion wird u.a. in einer VDI-Studie bestätigt (s. NETTESHEIM). In BUSCH/ENGELIEN/STAHN wurden Grundlagen, Anwendung und algorithmentheoretische Allgemeingültigkeit der Entscheidungstabellentechnik dargestellt. Durch ihre Begründung in der Allgemeinen Kybernetik besteht auch Gewißheit über die Allgemeingültigkeit der ARS-Technologie. Dennoch ist zu beachten, daß für spezielle Problemklassen und Applikationsbereiche spezielle Methoden, Techniken und Mittel auch ihre speziellen Vorzüge haben können oder eine effektive Lösung erst ermöglichen. Hierbei sollte man sich Gewißheit verschaffen, daß man die nutzbaren Vorzüge allgemeiner Vorgehensweisen auch erschöpfend nutzt.
Die Autoren wenden sich mit der ARS-Technologie an alle, die sich für Informationsverarbeitung und Steuerung interessieren, und besonders an jene, die an der Produktion von Software für automatisierte Informationssysteme aktiven Anteil nehmen. Mit dem in der deutschsprachigen Literatur (vgl. FLOYD/KOPETZ, KIMM/KOCH/SIMONSMEIER/TONTSCH, NETTESHEIM, ...) verbreiteten englischstämmigen Terminus Software-Engineering soll die Orientierung auf das Ingenieurwesen betont werden. Hierbei bedienen wir uns nach WUNSCH des Prinzips der Ingenieurwissenschaften, Erkenntnisse aus der Strukturzuordnung (Morphismenbetrachtung) zwischen technischen und mathematischen Bereichen der Kybernetik zu gewinnen, d.h., für ein technisches Problem suchen wir eine mathematische Abstraktion sowie eine Lösung in der Mathematik und versuchen dann, die abstrakte Lösung auf unser technisches Problem anzuwenden. Hierbei finden wir häufig auch nur teilweise passende Abstraktionen und Lösungen vor, die einer Modifikation und/oder Ergänzung bedürfen.

1.2. Gliederung und Hinweise zum Studium

Die in der Allgemeinen Kybernetik begründete Struktur der Darstellung der ARS-Technologie wird in Kap. 3. abgeleitet. Es sind dies die den Inhalt der Technolgie tragenden Kapitel 4., 5. und 6. sowie die Beziehungen zwischen ihnen.

Kap.1. EINFÜHRUNG beinhaltet eine Diskussion zu Gegenstand und Zielstellung der ARS-Technologie sowie eine Motivierung für die ARS-Technologie durch eine Analyse der sog. Softwarekrise.

Kap. 2. EINFACHER ABRISS ist ein Versuch, die ARS-Technologie i.e.S. ohne Verwendung komplizierterer mengentheoretisch-algebraischer Formalismen einzuführen und in der Anwendung darzustellen. Besondere Mühe wurde hierbei auf die Einführung und Anwendung der Kriterien für Qualität und Produktivität der Softwareproduktion verwendet. Wir hoffen, mit diesem Kapitel auch den weniger vorbereiteten Leser von der vorteilhaften Anwendbarkeit der ARS-Technologie zu überzeugen und soweit zu interessieren, daß er sich auch um die weiteren Kapitel bemüht. Der Preis für diese vereinfachte Darstellung mag an einigen Stellen von anderen Lesern als Langatmigkeit und terminologische Unschärfe empfunden werden. Das Kapitel schließt mit einer vergleichenden Darstellung verwandter Vorgehensweisen ab.

Kap. 3. FORMALISIERUNG UND GRUNDRISS dient der Darstellung ausgewählter allgemeiner Begriffe und Grundlagen der Kybernetik informationeller Prozesse, um unsere Position bezüglich in der Literatur auch unterschiedlich gebrauchter Begriffe zu fixieren, uns einige im weiteren tragende Begriffe, Vereinbarungen und Zusammenhänge zu vergegenwärtigen und die Struktur unserer Darstellung der ARS-Technologie abzuleiten.

Kap. 4. ELEMENTE beinhaltet die konstruktive Darstellung einfacher oder elementarer Prozesse der Informationsverarbeitung/ Steuerung, d.h. Algorithmierung oder Programmierung, mit den Mitteln der Automatentheorie, der nichtprozeduralen Entscheidungstabellentechnik und der prozeduralen Programmablaufplantechnik, die Bewertung von Qualität und Produktivität konstruktiver Darstellungen sowie eine Einführung in datenflußorientierte Darstellungen mittels PETRI-Netz-Technik.

Kap. 5. SYNCHRONISATIONSSTRUKTUREN beinhaltet die Vermittlung der ARS-Technologie im engeren Sinne. Für informationelle Prozesse, die eine horizontale allgemeine und rekursive Strukturierung zulassen, d.h. PETRI-Netze vom Typ Synchronisationsstruktur, werden die kybernetischen Grundlagen, ein ermitteltes System von Gesetzmäßigkeiten sowie ein abgeleitetes technologisches Diagramm zur Produktion von Software dargestellt und im

Beispiel demonstriert. Besonderer Wert wird hierbei auf die schrittweise Bewertung von Produktivitäts- und Qualitätsverbesserungen mit fortschreitender rekursiver Anwendung der ARS-Technologie bei der Projektierung gelegt. Wir sagen "ARS-Technologie im engeren Sinne", weil man sich im übrigen datenfluß-orientierten Software-Engineering meist ausschließlich dieser Art von Strukturierung widmet.

Kap. 6. KOORDINATIONSSTRUKTUREN beinhaltet Grundlagen der Koordinationstheorie, d.h. der vertikalen allgemeinen und rekursiven Strukturierung informationeller Prozesse, die Ableitung einer entsprechenden Methodik zur Projektierung koordinierter verteilter Steuerungen (meist schlicht nur hierarchische Steuerung genannt) sowie ihre Anwendung im Beispiel.

Kap. 7. ZUSAMMENFASSUNG beinhaltet schließlich eine zusammenfassende Diskussion der ARS-Technologie im weiteren Sinne, d.h. unter Einbeziehung des Stoffes der Kapitel 4., 5. und 6., Aussagen zur industriellen Anwendung und Rechnerunterstützung sowie einen Ausblick zum weiteren Ausbau der ARS-Technologie.

Wir hoffen, mit der gewählten Darstellung und Gliederung einen breiten Kreis von Lesern anzusprechen. In diesem Zusammenhang sind einige Teile auch autonom gestaltet, so daß man sich bei entsprechendem eingeschränkten Interesse vorerst auch nur Teilen widmen kann. Das betrifft insbesondere folgende Teile:

Kap. 2.- einfache Einführung in eine theoretisch begründete datenflußorientierte Softwaretechnologie im engeren Sinne;

Kap. 3., 4. und 5.- Formalisierung der ARS-Technologie als datenflußorientierte Softwaretechnologie im engeren Sinne,

Kap. 6.- Grundlagen und Methodik zur Projektierung koordinierter verteilter Steuerungen (hierarchische Steuerungen).

1.3. Softwarekrise und Schlußfolgerungen

Wir wissen, daß heute für Produktivitätssteigerungen in der Volkswirtschaft die Realisierung automatisierter Informationssysteme und Steuerungen als entscheidende technische Bedingung gilt. Nach BUDIG müssen etwa 24 bis 30 Prozent aller Hoch- und Fachschulkader der Bereiche Industrie, Landwirtschaft, Bauwesen und Verkehr Fachleute für Automatisierung sein. Die in diesen

Bereichen Studierenden müssen lernen, mit Mikrorechnern umzugehen. Diese erforderlichen Hoch- und Fachschulkader können nun nicht in wenigen Spezialrichtungen der Automatisierung ausgebildet werden. Es besteht die Notwendigkeit, in allen einschlägigen Fachrichtungen einen entsprechenden Anteil von Fachleuten verstärkt auf dem Gebiet der Automatisierung aus- und weiterzubilden. Folgend einer RGW-Nomenklatur, wollen wir diese Fachleute als "Automatisierungs-Spezialisten in den volkswirtschaftlichen Zweigen" bezeichnen.

Durch die Fortschritte der Mikorelektronik hat sich in den letzten zwanzig Jahren das Leistungs-Preis-Verhältnis für Hardware etwa um den Faktor 10^3 (für Prozessoren $> 10^3$, für Speicher $< 10^3$) verbessert, während sich die Produktivität der Softwareproduktion etwa nur um einen Faktor 3 - 5 verbesserte. Das hat dazu geführt, daß die Kosten für automatisierte Informationsverarbeitungssysteme und Steuerungen in zunehmendem Maße durch die Softwarekosten (heute 80 - 90 % Kostenanteil) bestimmt werden. Darüber hinaus wächst die installierte Hardwareleistung auf Mikrorechnerbasis derart, daß es unmöglich wird, die Software hierfür in der bisherigen Art und Weise bereitzustellen.

Nach VETTER(82) wird in den USA ein Computerzuwachs von 25 % pro Jahr beobachtet. NEMETH gibt für Ungarn eine entsprechende Zuwachsrate von 22 % an. Während in den USA nach VETTER(82) 1980 etwa 300.000 Programmierer tätig waren, würden bei nur üblichem Produktivitätszuwachs (5 - 6 % pro Jahr nach STETTER) in der Softwareproduktion 1990 in den USA etwa 28 Millionen qualifizierte Programmierer benötigt.

Diese sich verschärfende Diskrepanz zwischen Bereitstellung und Kosten für Hardware einerseits (viel und billig) und Software andererseits (zu wenig und zu teuer) wird international als Softwarekrise bezeichnet.

Die ARS-Technologie soll einen entscheidenden Beitrag zu ihrer Bewältigung leisten. Um diesen Beitrag werten und einordnen zu können, wollen wir einige weitere Erscheinungen der Softwarekrise darstellen und hieraus Schlußfolgerungen ableiten, die wir unter 1.1. bereits zu Aufgaben der ARS-Technologie erklärt haben.

Nach dem Gesagten sind die Hauptforderungen bereits offensichtlich. Produktivität und Qualität der Softwareproduktion sind entscheidend zu verbessern, wobei dem Einbeziehen von Speziali-

sten in den volkswirtschaftlichen Zweigen eine besondere Bedeutung zukommt, um automatisierte Informationssysteme in den volkswirtschaftlichen Zweigen im erforderlichen Umfange realisieren zu können.

Während der bisherige Produktivitätszuwachs in der Softwareproduktion nach STETTER und BALZERT hauptsächlich durch Verbesserungen bei den bereitgestellten Werkzeugen (z.B. Sprachen, Entwicklungssysteme, Arbeitsplatzterminale und -rechner, ...) und durch zunehmende Erfahrung gewonnen wurde, ist das im weiteren nicht mehr ausreichend, sondern nach STETTER: "Billigere Software hängt vor allem von einer höheren Produktivität des Entwicklungspersonals ab. ... Notwendig zur Verbilligung der Software ist vor allem das Verständnis des Entwicklungsprozesses." Nach BALZERT ergibt ein systematischer Vergleich von Methoden, Sprachen und Werkzeugen für die Projektierung von Software, daß es hierfür keine geschlossene Methode oder kein konsistentes Methodenbündel gibt, daß die Werkzeuge (zur Eingabe, Änderung, Dokumentation, Analyse und Simulation) zunehmend perfektioniert werden und daß bei den Sprachen graphische Netze (z.B. SADT, PETRI-Netze oder eben Datenflußstrukturen allgemein) den grundlegenden integralen Ansatz für die weitere Orientierung der Entwicklung darstellen.

Diesen Schlußfolgerungen zur Produktivitätssteigerung durch Gewinnung einer fundierten Transparenz des Erstellungsprozesses, durch interdisziplinäre Abstraktionen und graphische Netz-Darstellung der zu hantierenden Strukturen dient die Begründung der ARS-Technologie in der Allgemeinen Kybernetik. Es kann nach unserer Überzeugung nur auf dieser allgemeinen mathematischen Basis möglich sein, von der häufig als Schwarze Kunst bezeichneten Softwareprojektierungspraxis zu einer wissenschaftlich begründeten Technologie (ars nova) zu kommen. JONES formuliert hierzu: "Es ist die Mathematik, der man sich zuwenden muß, um eine vernünftige Basis für die Softwareprojektierung zu finden". KOPETZ formulierte zusammenfassend zur Tagung "Software Engineering - Entwurf und Spezifikation" (Berlin (West) 1980): "Die Diskussion hat gezeigt, daß wir noch sehr weit entfernt sind von einer einheitlichen und geschlossenen Methodologie der Softwareerstellung". Eine Diskussion auf dem IFIP-Kongreß 1983 soll lt. Ankündigung des Programmkomitees die Frage beantworten,

ob es sich bei den Projektierungstechniken für Software um Mythos, Zauberei oder reale Methodologie handelt.

VETTER(82) zeigt anhand einer IBM-Studie, daß sich die Entwicklungskosten zu den Durchführungskosten für Programme wie 129 : 1 verhalten, daß 50 % der existierenden Programme nur 2 % der Computerleistung in Anspruch nehmen und daß die mittlere Lebensdauer eines Programmes nur 15 bis 16 Monate beträgt. Bisher ging man hierfür in der Literatur von wesentlich anderen Werten aus. Damit ist aus unserer Sicht insbesondere die noch häufige Ansicht widerlegt, man brauche sophistische Programme (unter Ausnutzung von Nebeneffekten der Sprache, eines jeden Bits im Speicher usw.) und sophistische Programmierer würden durch formale Softwaretechnologien in ihrer Entfaltung gehemmt. Vielmehr ist Transparenz für einen breiten Kreis an der Softwareprojektierung Beteiligter notwendig, auch zu Lasten höherer Durchführungskosten. WENDT formuliert hierzu bezeichnend: "Der Kommunikationsansatz (sprich: Datenfluß-Netz-Technik) zielt auf die Beseitigung der Intransparenz, die dazu führt, daß sich Projekte nicht richtig planen lassen und dadurch unverhältnismäßig teuer werden. Außerdem können die gewonnenen Erfahrungen kein allgemein verfügbarer Besitz des Unternehmens werden, weil sie nur in den Köpfen Einzelner existieren. Für ein solches Informationsmonopol gilt - wie für jedes Monopol -, daß es von dem, der es hat, gehütet wird, während es Außenstehende verfluchen. Erst die Verbindung von Kommunikationsansatz und Analyseansatz kann die Software-Technik zu einer echten Ingenieurwissenschaft machen."

In engem Zusammenhang mit der Produktivität der Softwareproduktion steht also die Qualität der produzierten Software. Während bei gegebenem Produktivitätsniveau höhere Qualität mit höherem Aufwand verbunden ist, können auf höherem Produktivitätsniveau sowohl Qualität als auch Aufwand um Größenordnungen verbessert werden. Für die Diskussion hierzu ist es sinnvoll, ein in der Literatur übliches "Lebensphasenmodell" für Software einzuführen. Wir beschränken uns auf die in allen derartigen Modellvarianten erkennbaren Hauptphasen, die in etwa auch denen entsprechender Phasenmodelle aus anderen Technikbereichen entsprechen (vgl. z.B. KLÖPPEL/BUTZ/WINOSLAWSKIJ/PRACHOWNIK):

- Planung/Projektierung mit dem Plan z.B. als Datenflußstruktur,
- Ausführung/Konstruktion mit dem Konstrukt z.B. als Programm,
- Nutzung des implementierten und übergebenen Programmes, ggf. Änderung.

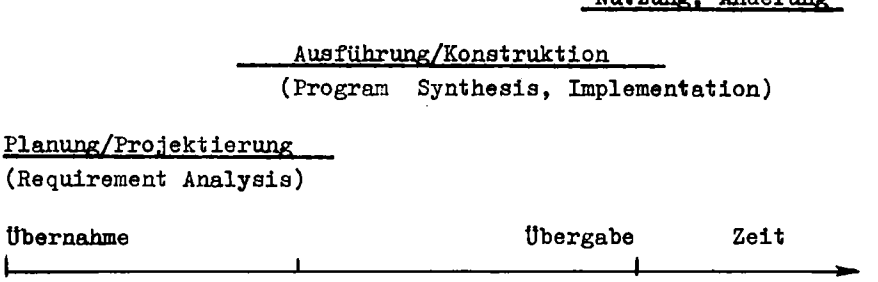


Abb. 1 Phasenmodell Softwareproduktion (eines Auftrages)

Nach SCHNEIDER zeigen sich bei großen Betriebssystemen und Datenbanksystemen 500 bis 1000 Fehler in der Nutzung. Nach CONNOR wird die Hälfte aller Fehler im Leben eines Softwareproduktes erst bei oder nach seiner Übergabe entdeckt, wobei hiervon etwa 80 % bereits vor der Programmierung, d.h. im Laufe der Projektierung, verursacht werden. Ferner ist es nach VETTER(82) so, daß Fehler um so später entdeckt werden, je früher sie verursacht werden, und daß Fehlerkorrekturen um so teurer sind, je später sie gemacht werden (können).
BEICHTER vermerkt hierzu ergänzend, daß 80 % der Kosten für Fehlerbeseitigung auf Entwurfsfehler entfallen.

Aus diesen Erkenntnissen folgt, daß die Qualifikation der Projektierungsphase entscheidend für die Qualifikation der gesamten Softwareproduktion ist. Hier werden Qualität und Kosten weitestgehend bestimmt. Das klingt in einer Kritik an Entwurfsverfahren im Hinblick auf Qualitätsanforderungen bei Realzeit-Software nach BELLI wie folgt: "Die Entwurfsphase ist wegen ihrer möglichen schwerwiegenden Folgen auf die weiteren Phasen des Lebenszyklus eines Softwareproduktes die wichtigste Phase. Die entscheidenden Fragen dieser Phase lauten:
- Wie können bereits zu diesem Zeitpunkt die Qualitätsmerkmale berücksichtigt werden?

- Wie kann eine wirtschaftliche Generierung von Testdaten garantiert werden?"

Eine Qualifikation der Softwareprojektierung und damit der -produktion ist also durch mathematisch fundierte Methoden, durch interdisziplinäre Zusammenarbeit von Softwarespezialisten mit Automatisierungsspezialisten bei Anwendern und Auftraggebern sowie durch eine Qualitätsbewertung bereits in der Projektierungsphase zu erreichen. Durch Verlagerung von Aufwand aus der Konstruktionsphase in die Projektierungsphase kann das Aufwandsintegral für ein Softwareprodukt verringert werden und seine Qualität (z.B. auch die Fehlerfreiheit) verbessert werden, was nach VETTER(80) in Abb.2 illustriert wird. Die ARS-Technologie fordert und bietet eine diesen Forderungen entsprechende neue Vorgehensweise, insbesondere für die Projektierung. Nach WASSERMAN/GUTZ muß an die Stelle der heutigen Anwendungsprogrammierung in Zukunft die Projektierung (specification, requirement analysis) treten.

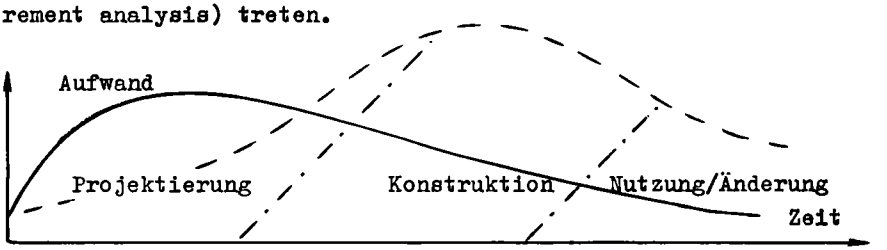


Abb. 2 Aufwandsverlagerung und -verringerung in der Software-Produktion (— — — — ⟶ ————)

Im Zusammenhang mit den zu Abb. 1 und Abb. 2 formulierten Aussagen läßt ein Software-Phasen-Modell nach WEDEKIND eine weitere wichtige Aussage anschaulich werden. Für viele Software-Erstellungen ist es heute im einzelnen prinzipiell noch möglich, in konventioneller Weise lediglich über die "Konstruktions-Ebene" (s. Abb. 3) eine Lösung zu erstellen, d.h. ohne explizite Nutzung der "Schema-Ebene" (Projektierung). Dieser im einzelnen gangbare Weg wird von sophistischen Programmierern als Argument gegen die Notwendigkeit zur Begehung der "Schema-Ebene" gebraucht. Wir möchten hier deutlich machen, daß die Softwarekrise wesentlich durch eine mangelnde Qualifikation und Durchsetzung der Projektierungsphase ("Schema-Ebene") bedingt ist und eine moderne industrielle Softwareproduktion der Projektie-

rungsphase bedarf. Auch in NETTE SHEIM ist hierzu formuliert: "Für die Ausführung eines Projektes ist ein Plan unentbehrlich, der die Struktur und die wesentlichen Merkmale allgemein verständlich darstellt".

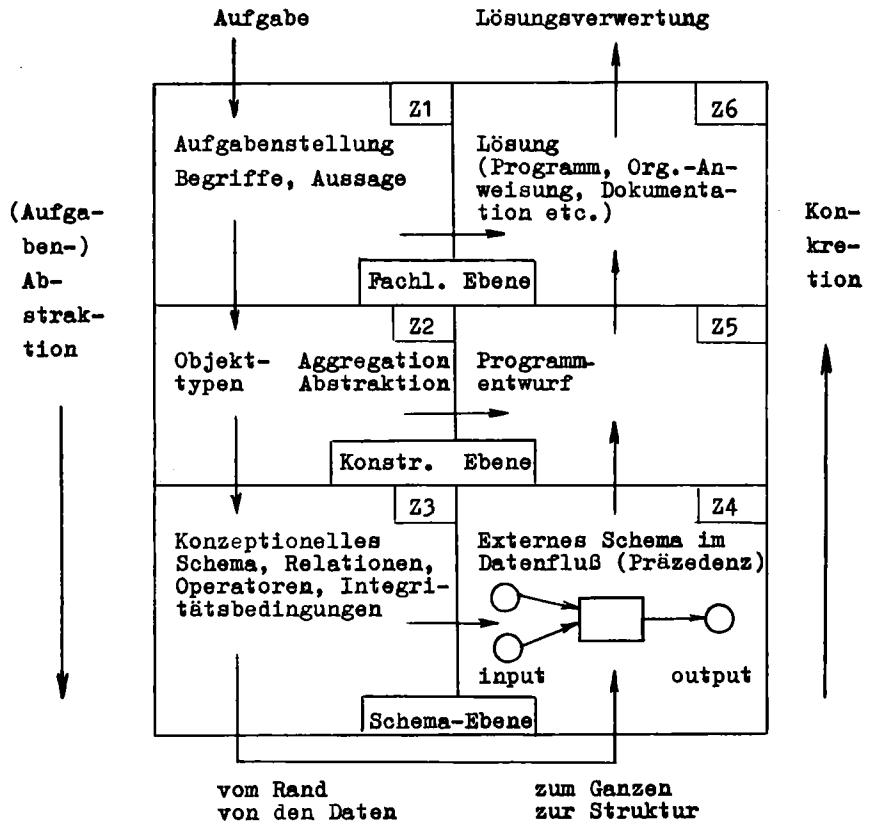


Abb. 3 Software-Phasen-Modell von WEDEKIND (nach LAUCKS/OTTE/VERMEIREN)

Weitere Probleme der Softwareproduktion sind die der Dokumentation und Wartung, die eng miteinander verflochten sind. Nun wird zwar für Softwareprodukte Dokumentation gefordert, jedoch ist diese meist ungeeignet, als Grundlage für Reproduktionen, Korrekturen, Änderungen usw. zu dienen. Das liegt vorwiegend daran, daß einerseits bei der steuerflußorientierten Vorgehensweise es kein explizites und dokumentierfähiges Zwischenprodukt (es existiert höchstens implizit im Kopfe des Programmierers) von der

Projektierung zur Konstruktion gibt und daß andererseits selbst der Konstruktionsentwurf Programmablaufplan als erstes dokumentierfähiges Zwischenprodukt überhaupt bei der Implementierung solche Veränderungen erfährt, die sich der Dokumentation entziehen. EHLING formuliert hierzu: "Nun erarbeitet ein Programmierer sehr wohl Entwurfsdokumente, aber - ungleich den Bauplänen usw. - wirft er sie in den Papierkorb, nachdem er mehr oder weniger glücklich im Programm "gelandet" ist. Warum? Sie stimmen nicht mehr. Das heißt: sie haben nie gestimmt."

Folgen hiervon sind ein geringer Nachnutzungsgrad von Softwareprojekten (80 - 90 % aller Programme doubeln sich gegenseitig - STOGNI)- da das Programm bei nur gering abweichenden Nutzungsbedingungen (Betriebssystem, Peripherie, ...) nicht portabel ist und Zwischenprodukte nicht existieren oder unbrauchbar sind- und ein hoher Anteil (etwa 2/3 heute, steigend) der Wartung (Korrektur, Änderung, ...) an den Gesamtaufwendungen für Software. Nach STETTER ist die Softwarewartung nach wirtschaftlichen Maßstäben das Problem für die Softwaretechnologie. Wartung eines fertigen Programmsystems bedeutet Beseitigung von noch erkannten Fehlern und Anpassung an sich ändernde Nutzungs- und Hardwarebedingungen. In einer Zusammenfassung zur IFIP-Konferenz über Evolutionäre Informationssysteme (Budapest 1981, s. HAWGOOD), d.h. an sich ändernde Nutzungs- und Anforderungsbedingungen anpaßbare Informationssysteme, formulierte VERRIJN-STUART (Chairman): "Der Mangel an einer soliden theoretischen Basis ist offensichtlich". Die in den Beiträgen erkennbaren formalen Ansätze waren mit PETRI-Netz, Transitions-Diagramm, SADT-Methode sowie der ABRAHAM-Methode auf ISAC-Basis im Hauptvortrag von SCHEIDER datenflußorientiert. SCHOPPAN gibt für den RGW-Bereich auf bestimmte Arten von Ursachen entfallende Anteile von Softwareänderungen an: 30 % auf Entwurfsfehler, 6 % auf Programmfehler, 30 % auf Systemumstellungen, 20 % auf interne Verbesserungen, 9 % auf Anpassung durch Nachnutzer und 5 % auf Veränderung gesetzlicher Bestimmungen.

Die ARS-Technologie bietet insbesondere mit ihrem Projektierungsergebnis in Form einer datenflußorientierten Prozeßstruktur auf der Basis allgemeiner Gesetzmäßigkeiten informationeller Transformationen ein explizites, verifizierbares, bewertbares, portables, änderbares und dokumentierbares Zwischenergebnis,

welches sich in den bisherigen Anwendungen der ARS-Technologie insbesondere auch als evolutionär in oben genanntem Sinne einer mit sich ändernden Bedingungen mitwachsenden Struktur bewährt hat.

Wir haben mit dieser einleitenden Darstellung zur Softwarekrise versucht, durch Fakten, Zusammenhänge und Aussagen internationaler Experten Notwendigkeit, Anlage und Einordnung der ARS-Technologie darzustellen, um auch pessimistische Leser anzureizen, denn auch wir haben vielfach dem nachfolgenden Zitat nach LESSHAFFT ähnelnde Erfahrungen gemacht, ohne uns jedoch dem pessimistischen Schluß anzuschließen:

"Hält man sich ferner vor Augen, wie weit die Praxis der Software-Entwicklung in normalen Anwendungsinstallationen noch von der an den Universitäten und den großen Softwarelaboratorien geführten Methodendiskussion entfernt ist, das heißt in welcher methodischen Vorsintflutlichkeit gegenwärtig noch die normale Anwendungsprogrammierung versinkt, dann wird klar, daß die Forderung nach theoriebewußter Systemanalyse bis auf weiteres der Banalität des Zeit- und Kostendrucks, der Wartungsprobleme und der Ausbildungsmängel zum Opfer fallen wird. Die Zeit ist noch nicht reif für Optimismus."

Wir möchten uns mit unserem Anliegen zur Vermittlung der ARS-Technologie der zusammenfassenden Erkenntnis von WENDT anschliessen:

"Es ist allgemein bekannt und wird in Diskussionen, die softwaretechnische Probleme betreffen, immer wieder ausgesprochen, daß auf vielen technischen und wissenschaftlichen Gebieten entscheidende Fortschritte erst nach Entwicklung adäquater und übersichtlicher Darstellungsformen möglich wurden und daß in der Software-Technik die gleiche Abhängigkeit zwischen der Entwicklung der Darstellungsverfahren und entscheidenden technischen Fortschritten besteht. Diese Erkenntnis ist Ausgangspunkt des Kommunikationsansatzes." (Kommunikationsansatz heißt Anwendung datenflußorientierter Methoden wie Instanzen-Netz-, PETRI-Netz- oder SADT-Methode.) "Die Entwicklung der Software-Technik zu einer soliden Ingenieurwissenschaft ist zwar noch nicht abgeschlossen, aber man ist dem Ziel deutlich näher gekommen."

In ähnlicher Weise sieht KRÖGER auch 1983 den entscheidenden

Schritt zur Überwindung der Softwarekrise in der Durchsetzung graphisch orientierter Analysemethoden auf der Basis von Systemdenken im Gegensatz zum verbreiteten Denken des Analytikers in steuerflußorientierten Ablaufstrukturen. Nach seiner Meinung war und ist es der Mangel an Transparenz in der Darstellung, der es nicht zuläßt, Mißverständnisse zwischen Anwender und Analytiker frühzeitig zu erkennen und zu klären. Bezüglich der Relation zwischen den drei wichtigsten Einflußgrößen der Software-Erstellung Tools, Methoden (besonders Analysemethoden) und Personen schlußfolgert er folgendes:
"An erster Stelle steht die Einstellung des Analytikers zu sich selbst und zu den anderen Projektbeteiligten - und da besonders zum Anwender. Es folgt die Beherrschung von Methoden, besonders von Analyse- und Entwurfsmethoden, deren "bildnerische" Darstellungskraft mehr Verständigung und damit eine Unterstützung der "Einstellung" erreicht. Erst dann kommen die Werkzeuge (Tools) zur Unterstützung der Projektarbeit."

Die Bedeutung der Einstellung zu den neuen Methoden für deren Durchsetzung macht folgende Schlußfolgerung einer Studie von LAUCKS/OTTE/VERMEIREN deutlich: "Mit den in der bisherigen Praxis vorhandenen Methoden kann zwar jedes Software-Projekt als Einzelproblem, aber nicht das Problem in seiner Gesamtheit gelöst werden."

2. EINFACHER ABRISS

2.1. Gegenstand und Vereinbarungen

In diesem Kapitel wollen wir die Grundidee der ARS-Technologie i.e.S., d.h. bei nur horizontaler allgemeiner Strukturierung, ohne Verwendung komplizierterer mengentheoretisch-algebraischer Formalismen einführen, um vor allem auch den diesbezüglich weniger vorbereiteten Leser zu interessieren.
Die Durchsetzung der horizontalen allgemeinen Strukturierung hat für die breite Produktion von Anwendungssoftware die größte Bedeutung. In den zum Schluß dieses Kapitels vorgestellten Vorgehensweisen der Softwarefirmen IBM, SofTech Inc. und ACTIS GmbH wird ausschließlich sie eingesetzt. Die vertikale allgemeine Strukturierung hat ihren vorrangigen Platz beim Entwurf hierarchischer verteilter Steuerungen.
Der Verzicht auf eine formalisierte und damit schärfere Darstellung in diesem Kapitel basiert auf eigenen und in der Literatur dargestellten Erfahrungen, daß der z.B. mit WUNSCH/SCHREIBER angestrebte Vorbereitungsstand anzusprechender Leser sich erst langsam auszuprägen beginnt. Vermutlich wird aus diesem Grunde auch im IBM-Kurs "Methoden zur Gestaltung von Informationssystemen" nach VETTER vorwiegend mit verbalen und graphischen Darstellungen ohne kompliziertere mengentheoretisch-algebraische Formalismen gearbeitet. Ähnliches trifft für die bekannten Darstellungen anderer Softwaretechnologien zu.

Wir beginnen mit der Fixierung des Gegenstandes unserer Diskussion, wofür wir einige einfache Formalismen einführen.
Es folgt unter 2.2. eine Betrachtung von Kriterien zur späteren Bewertung der Effektivität unserer Vorgehensweisen bezüglich einer einfacheren Erstellung besserer Software, wofür wir uns vorwiegend des allgemein verbreiteten Formalismus der Programmablaufpläne bedienen.
Dann führen wir unter 2.3. die horizontale allgemeine Strukturierung ein und formalisieren sie mittels graphischer Darstellungen und verbaler Regeln.
Unter 2.4. fassen wir die Ergebnisse zur Strukturfindung zusammen, ergänzen die somit diskutierte Projektierungsphase durch Aussagen zur Konstruktionsphase und geben ein verbal-graphisches Schema zur ARS-Technologie i.S. dieses Kapitels an.

Auf dieser Basis diskutieren wir unter 2.5. ein Beispiel zur Anwendung des technologischen Schemas, wobei wir unseren Fortschritt mittels Ausprägung der Kriterien nach 2.2. bewerten. Schließlich diskutieren wir unter 2.6. zum Vergleich einige verwandte Vorgehensweisen.

Startpunkt für die Produktion einer Software ist eine Aufgabenstellung/Anforderung zur Realisierung eines Prozesses der Informationsverarbeitung oder Steuerung: Geforderte Ausgangsinformation (Output) ist mittels algorithmischer bzw. programmtechnischer Transformation (Processing) von Eingangsinformation (Input) zu ermitteln.
In der Praxis (vgl. IPO-Technik) ist hierfür bereits die Notation O = P(I) oder P: I → O gebräuchlich (I - Input, P - Process, O - Output). Um eine Entsprechung zur Theorie zu erhalten, wollen wir folgendes vereinbaren:

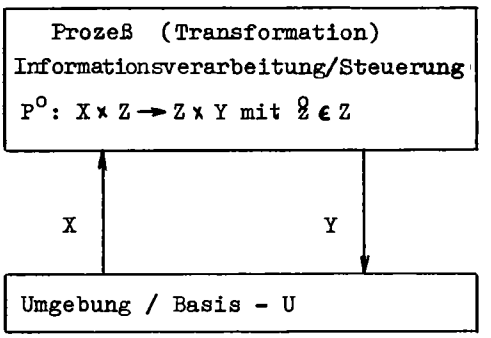


Abb. 1 X-Y-Kommunikation von P^o-U

Träger der Eingangsinformation sind Eingangsgrößen $x \in X$. Träger der Ausgangsinformation sind Ausgangsgrößen $y \in Y$. Ein Wert, z.B. $x(t) \in X$, kann eine Schalterstellung, ein Meßwert, eine Meldung usw. sein. Eine zeitliche Folge von Werten einer Größe, z.B. $x(1)x(2)...x(t)...x(w) = \underline{x} \in \underline{X}$, bzw. auch die Menge aller solcher Folgen wollen wir durch Unterstreichung des Namens der entsprechenden Größe bzw. ihres Wertevorrates bezeichnen. Dann bezeichnet $\underline{P}^o$: $\underline{X} \rightarrow \underline{Y}$ bei fixiertem Anfangszustand $z_0 \in Z$ einen Transformationsprozeß Informationsverarbeitung/Steuerung oder $\underline{y} = \underline{P}^o(\underline{x})$ die Lösung einer Aufgabe dieses Prozesses. Eine derartige Transformation von Information erfolgt taktweise zu definierten Zeitpunkten $t \in T = \{1,2,...,w\}$. Zu jedem Zeitpunkt können wir von außen Werte $x(t) \in X$ und $y(t) \in Y$ sowie ihre Zuordnung $x \mapsto y \in X \times Y$ beobachten. Der Transformationsprozeß befindet sich zu jedem Zeitpunkt $t \in T$ in einem bestimmten Zustand $z(t) \in Z$. Unter Verwendung der Zustandsgröße $z \in Z$ läßt sich der Transformationsprozeß einfacher (ohne explizite Hantierung von Folgen)

wie folgt notieren:

$$P^o: X \times Z \rightarrow Z \times Y \quad \text{mit } z(t=0) = \underset{\sim}{z} \in Z.$$

Wir merken uns also eine "Vorgeschichte" mittels Zustandswert und ermitteln $y(t) = (x(t), z(t-1))$ und $z(t) = (x(t), z(t-1))$ zu jedem Taktzeitpunkt $t \in T$, so daß sowohl P^o wie auch $\underline{P}^o$ für jede durch eine Folge $\underline{x}$ definierte Aufgabe die gleiche Lösung $\underline{y}$ bei gegebenem Anfangszustand $\underset{\sim}{z}$ liefern.
Eine Transformation P^o kann mehrere Eingangs-, Ausgangs- und Zustandsgrößen haben. Das schreiben wir dann als Mengenprodukt:

$$X = X1 \times X2 \times \ldots, \quad Y = Y1 \times Y2 \times \ldots, \quad Z = Z1 \times Z2 \times \ldots .$$

Eine Transformation P^o erfolgt nach einer informationellen Vorschrift, die wir Algorithmus nennen, auf einer physischen Einrichtung, die wir Prozessor nennen. Wenn der Algorithmus in einer Programmiersprache für einen automatisierten Prozessor (Rechner, Computer) dargestellt ist, nennen wir ihn Programm, welches wiederum in Einheit mit seiner Dokumentation und weiteren Ergänzungen Software darstellt.

Ziel der Softwareproduktion ist somit die Erstellung eines Algorithmus zur Realisierung der geforderten Transformation $\underline{P}^o$, wobei ein bestes Ziel mit geringstem Aufwand erreicht werden soll. Ein Algorithmus seinerseits ist ein informationelles System, welches sich insbesondere durch seine Struktur auszeichnet. Diese Struktur gilt es zu finden.
Hierfür gehen wir von folgenden Grundgedanken aus:

- Eine Struktur ist eine Komposition von Elementen. Somit finden wir eine Struktur mittels Dekomposition eines angenommenen Ganzen.
- Die Struktur Algorithmus soll eine geforderte informationelle Transformation $\underline{P}^o$ realisieren. Wenn diese Transformation eine Struktur hat, ist es naheliegend, die Algorithmenstruktur "ähnlich" zur Transformationsstruktur zu komponieren.

ARS bedeutet "Allgemeine und Rekursive Strukturierung" und ist eine Vorgehensweise zur Dekomposition einer informationellen Transformation der Art $\underline{P}^o: \underline{X} \rightarrow \underline{Y}$ mit dem Ziel einer effektiven Komposition eines realisierenden Algorithmus. Bei der Strukturfindung für eine konkrete Transformation $\underline{P}^o$ sowie bei der Diskussion unserer Vorgehensweise unter 2.3., aber auch bereits für die Diskussion der Bewertung unter 2.2. gehen wir von den "von

außen beobachtbaren" Merkmalen X und Y aus, wobei häufig X auch erst aus Y per Aufgabenstellung zu $\underline{P}^{o}$ abgeleitet werden muß. Diese Vorgehensweise zur Softwarestrukturierung über den scheinbaren Umweg der Prozeßstruktur (vgl. Abb. 3/1.3.) wird für viele neu sein. Sie ist jedoch theoretisch begründet und bewährt sich bereits in bedeutenden Bereichen der Praxis.

2.2. Bewertung von Produktivität und Qualität

Nun genügt uns nicht, einen Anspruch auf effektive Vorgehensweise zu erheben, sondern wir wollen auch die Frage nach der Bewertung der Effektivität beantworten. In der Literatur findet man eine Reihe von Vorschlägen für Bewertungskriterien, wobei in den letzten Jahren z.B. für die Qualität neben den traditionellen quantitativen Kriterien wie Speicherplatz und Rechenzeit immer mehr auch qualitative Kriterien wie Wohlstrukturiertheit, Lesbarkeit, Änderbarkeit usw. aktuell geworden sind. Weiterhin steht für die Projektierung im speziellen wie auch für das Software-Engineering im allgemeinen die Forderung, notwendigen Aufwand und erreichbare Qualität bereits zu bewerten, bevor ein Endprodukt vorliegt. Das ist wichtig für die Planung der Softwareproduktion, für die Bewertung verschiedener Vorgehensweisen, meist als Technologien bezeichnet, sowie für die Bewertung einzelner Projektierungsschritte bei gegebener Technologie.

Wir führen unter diesen Gesichtspunkten folgende Kriterien zur Bewertung bei der Softwareproduktion ein:

- für den Aufwand (umgekehrt proportional zur Produktivität bei fixierter Zeit)

o - nominale Anzahl von funktionell äquivalenten Programmablaufplänen für Auswahl eines optimalen,

\+ - nominale Anzahl von Situationen für vollständigen Funktionstest des Algorithmus;

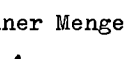
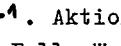
- für die Komplexität (umgekehrt proportional zur Qualität)

x - nominale Anzahl von Testknoten eines Programmablaufplanes,

" - Anzahl informationeller Abhängigkeiten zwischen Algorithmeneinheiten (Parallelitäts-Komplement).

Nun ist es einerseits angenehm, daß es sich hier um quantitative Kriterien handelt. Das ist für die Effizienzbewertung normal. Es bleibt aber andererseits die Frage: Wie steht es um die Bewertung von Transparenz, d.h. Wohlstrukturiertheit, Lesbarkeit, Änderbarkeit usw.? Fundierte Untersuchungen, s. z.B. WOODWARD/HENNEL/HEDLEY, haben ergeben, daß das oben eingeführte Kriterium Testknoten-Anzahl gut korreliert mit dem zentralen und modernen qualitativen Kriterium Wohlstrukturiertheit i.S. von DIJKSTRA (somit auch mit den Kriterien Lesbarkeit, Änderbarkeit usw.), aber auch mit weiteren Qualitätskriterien, wie z.B. Softwarekosten, Fehlerfreiheit, Anzahl der Anweisungen (somit auch Speicherplatz und Rechenzeit) u.a.m. Im weiteren werden wir auch gute Korrelation zu unseren anderen drei Merkmalen zeigen, d.h. eine gute gegenseitige Korrelation aller vier.

Um mit den eingeführten Bewertungskriterien hantieren zu können, erläutern wir sie zuerst insbesondere bezüglich ihrer Wertausprägung.
Der Programmablaufplan (kurz: PAP; allgemein auch als Ablauf-Graph bezeichnet, vgl. KAMMERER) der konventionellen Softwareproduktion kann als sprachunabhängige strukturelle Abstraktion eines Softwareproduktes "Programm" betrachtet werden. In dieser Eigenschaft wollen wir ihn für unsere Bewertungen anstellen von in speziellen Programmiersprachen realisierbaren Softwareprodukten betrachten.

Ein PAP ist eine mit Wahrheitswerten $\{0,1\}$ bewertete Struktur auf einer Menge von Testknoten  und Aktionsknoten  . Aktionen als Inhalte von Aktionsknoten sind im einfachen Falle Wertzuweisungen für solche Größen, die im rechten Teil der Transformation P^0: $X \times Z \rightarrow Z \times Y$ vorkommen, d.h. für Faktoren An aus dem Produkt $Z \times Y = A1 \times A2 \times \ldots \times An \times \ldots = \overset{\cdot}{\underset{\cdot}{\times}} An$ (z.B. a3:=(b1-b3)*4).
Bedingungen als Inhalte von Testknoten (Steuerflußverzweigungen) sind Aussageformverbindungen über Größen, die im linken Teil der Transformation P^0: $X \times Z \rightarrow Z \times Y$ vorkommen, d.h. über Faktoren Bm aus dem Produkt $X \times Z = B1 \times B2 \times \ldots \times Bm \times \ldots = \overset{\cdot}{\underset{\cdot}{\times}} Bm$ (z.B. b2=0 oder b3 > b1+b2). Die mit "1" bewerteten Ausgangskanten werden bei erfüllter Aussageformverbindung (wahrer Aussage) verfolgt (Steuerfluß). Ein und dieselbe Bedingung kann in verschiedenen