Die Interessengruppe Programmpakete

PETER BACHMANN
bachmann@informatik.tu-cottbus.de

In der Interessengruppe Programmpakete wurden die intensiven Diskussionen zu Programmsystemen in einem Buch zusammen gefasst. Unterdessen sind weltweit unter dem Thema „Softwaretechnik" vielfältige Untersuchungen durchgeführt worden, die die Inhalte des Buches berühren. In diesem Artikel werden einige der betrachteten Aspekte nochmals beleuchtet und mit den neueren Ergebnissen verglichen.

1 Die Geschichte

1976 fand sich eine Gruppe von Fachkollegen aus dem Bereich der Akademie der Wissenschaften, dem Hochschulwesen und der Industrie zusammen, um den wissenschaftlichen Informationsaustausch zur Fundierung, Entwicklung und Anwendung von Programmsystemen zu pflegen. Man gab sich den Namen „Interessengruppe Programmpakete". Damit sollte ausgedrückt werden, dass das Hauptmotiv das wissenschaftliche Interesse am Gegenstand ist. Die Verwertung eventueller Resultate stand nicht im Focus der Diskussion.

Anfangs divergierten die einzelnen Ansichten, was insbesondere im unterschiedlichen Sprachgebrauch lag. Bald wurde erkannt, dass die Erarbeitung eines einheitlichen Begriffssystems zur Darstellung und Bewertung der verschiedenen Positionen unumgänglich ist. Im Zuge dieser Arbeit näherten sich dann die vertretenen Auffassungen zur Nutzung, der Architektur, der Steuerung, der Entwicklung und der Modellierung an. Allerdings war man gegenseitig gern zu Eingeständnissen und Kompromissen in den Ansichten bereit, solange dies nicht mit Konsequenzen verbunden war.

Um den in den Diskussionen erreichten Standpunkten eine größere Verbindlichkeit zu geben wurde als gemeinsames Ziel deren Herausgabe in Buchform vereinbart. Von da an gestaltete sich der wissenschaftliche Meinungsstreit konsequenter, aber auch effektiver.

Nach sechs Jahren intensiver gemeinsamer Arbeit entstand das Manuskript, das 1983 unter dem Titel „Programmsysteme: Anwendung – Entwicklung – Fundierung“ im Akademie-Verlag Berlin erschien [2] und an dem die elf Mitglieder der Interessengruppe

Peter Bachmann, Dresden (Leiter und Herausgeber),
Martin Frenzel, Dresden,
Frank Heltzig, Dresden,
Dieter Herrig, Schwerin,
Karl-Heinz Kutschke, Rostock,
Walter Mach, Karl-Marx-Stadt,
Horst Sandmann, Berlin,
Kuno Schmidt, Berlin,
Michael Schwaar, Karl-Marx-Stadt, und
Horst Sobotta, Dresden

als Autoren mitwirkten.

Neben den oben bereits erwähnten fünf Aspekten Nutzung, Architektur, Steuerung, Entwicklung und Modellierung besteht das Buch aus einem einleitenden Kapitel zur allgemeinen Einordnung von Programmsystemen und dem Lösungsprozess sowie einem abschließenden Kapitel, in dem sieben konkrete Programmsysteme nach einheitlichen Gesichtspunkten vorgestellt sind.

Im folgenden soll nun betrachtet werden, inwieweit sich die im Buch vertretenen Standpunkte in den vergangenen 30 Jahren bestätigt haben, was davon vielleicht nur unter einer neuen Bezeichnung neu eingeführt wurde und welche grundsätzlichen Aspekte neu entwickelt wurden.

Dabei muss ich selbstkritisch einräumen, dass ich mich hauptsächlich auf die angegebenen Literatur [4, 8] sowie auf das Gehörte und Erlebte während meiner Tätigkeit an der BTU Cottbus, Institut für Informatik, gestützt habe. Persönlich habe ich mich nach 1982 nicht mehr zum globalen Gegenstand, sondern nur zu speziellen Fragen wie zum Beispiel der Verifikation [3] forschungsmäßig betätigt.

2 Das Grundkonzept

Da die elf Mitglieder der Interessengruppe hauptsächlich aus dem Bereich Wissenschaft und Technik kamen, standen derartige Anwendungen im Zentrum der Betrachtungen. Auf diesem Gebiet gab es viele persönliche Erfahrungen durch die Entwicklung von Pilotprojekten. Sieben davon sind im letzten Kapitel des Buches beschrieben, wobei eine einheitliche Beschreibungsform angestrebt wurde. Bei Programmsystemen zur Ökonomie bzw. Verwaltung gab es weder eigene Entwicklungen noch ein entsprechendes Interesse.

Als Konsequenz wurden **funktionelle und algorithmische Aspekte** besonders intensiv behandelt, was unter anderem bei den Betrachtungen zur Steuerung zum Ausdruck kam. Auch waren die Pilotprojekte von relativ kleinen Entwicklerkollektiven (heute: Teams) erarbeitet worden, wodurch der Entwicklungsprozess und das Projektmanagement keine große Aufmerksamkeit erhielt. Dagegen war uns die ausführliche **Diskussion von Gestaltungsprinzipien** wichtig. Dementsprechend wurde auch das Buch gegliedert. In [4, 8] wird dagegen entsprechend der Entwicklungsphasen vorgegangen.

Die Informatik in der DDR wurde entscheidend durch N. J. Lehmann mit geprägt. Neben vielen anderen Gebieten in der Informatik hatte sich Lehmann insbesondere für eine „sprachlich geführte Programmiertechnologie“ eingesetzt. Er war es auch, der im TEKMO-Seminar die Beziehung zwischen Programmpaket und Sprachen in das Zentrum der Aufmerksamkeit rückte (Siehe [7, 9]). Es war also natürlich, dass die Interessengruppe von diesem Konzept ausging. So wird auch im Buch an vielen Stellen auf die **Bedeutung der Sprachen** eingegangen.

Anfang der siebziger Jahre war die strukturierte Programmierung als möglicher Ausweg aus der Softwarekrise ein stark beachteter Gegenstand, durch die auch der Begriff des Moduls geprägt wurde. Wirth entwickelte mit ModulaII eine Sprache, in der Module das zentrale Element darstellten. Auch die Interessengruppe hat **Modularisierung** als wichtiges Gestaltungsprinzip herausgestellt. Da die Objektorientierung erst später ihren fast euphorischen Höhepunkt erreichte, kam dies in den Diskussionen der Interessengruppe nicht vor.

3 Programmsysteme und Lösungsprozess

In diesem einleitenden Kapitel wird versucht, aus den Untersuchungen zum Lösungsprozess als Arbeitsprozess Anforderungen an die Gestaltung von Programmsystemen abzuleiten.

Am Anfang unserer Diskussion waren viele Mitglieder der Interessengruppe skeptisch, ob eine solche Betrachtung wirkungsvoll ist. Hauptsächlich durch Dieter Herrig wurden wir hinreichend überzeugt, dass dies möglich und sinnvoll ist.

Ausgangspunkt ist, dass informationelle Arbeitsprozesse

- als Arbeitsgegenstände Texte (Schrifttexte, Grafiktexte) haben,
- als Arbeitsmittel Hardware und Software haben und
- Lösungsprozesse (Aufgabenlösungsprozesse oder Problemlösungsprozesse) sind.

Dass Texte hier so hervorgehoben werden, entspricht dem Grundkonzept. Der Begriff „Text“ wurde sehr weit gefasst, es wurden auch Grafiken und Dia-

gramme einbezogen (fraglich, aber für uns praktisch), also waren Texte allgemein Elemente von Sprachen. Allgemein erwartet und gebräuchlicher ist der Begriff der „Software“, der im Laufe der Zeit viele Definitionen erfahren hat. Es gab ja auch eine Zeit, in der in der DDR diese Bezeichnung auf Grund ihrer ideologischen Verbrämung durch „Systemunterlagen“ ersetzt wurde. Wir hatten uns dem nicht angeschlossen, aber auch eine Definition vermieden. Stattdessen wurde leger formuliert:

> „Wir verstehen unter Software (im weiten Sinne) alle Texte, die den potentiell universellen Rechner aktuell spezifizieren“.

Eine solche Auslegung habe ich in der modernen Literatur nicht gefunden. Dort bezieht man sich meist auf den Begriff „Programm“, der allerdings kaum besser definiert ist.

Die Arbeitsmittel in Form von Hardware und Software ist in Rahmen und Basen unterteilt. **Rahmen** *organisieren* den Lösungsprozess und bestehen aus:

- Lösungsvorschriften als Szenarien (Szenenbasis),
- Lösungsschritte als Programme (Programmbasis) und
- Lösungselemente als Daten (Datenbasis).

Rahmen und Basen werden in verschiedenen **Ebenen** (Abstraktionsstufen) entworfen, nämlich:

- in der externen Ebene des Fach-Praktikers,
- in der konzeptuellen Ebene des Fach- und EDV-Theoretikers und
- in der internen Ebene des EDV-Praktikers.

Jede Ebene wird durch eine **Schicht** realisiert, die auf die Mittel der darunter liegenden Schicht Bezug nimmt. **Kerne** sind formbezogene Teile (z. B. Grafikkerne, Kommunikationskerne).

An dieser (sehr) allgemeinen Betrachtungsweise orientiert sich nun das weitere **Vorgehen**: um ein Programmsystem zu entwickeln sind, nach Bestimmung der Aufgabenstellung,

- Rahmen auszuwählen bzw. zu entwickeln und
- Basen zu füllen, indem
  - Szenarien für die Lösungsfindung definiert werden,
  - Programme für die einzelnen Lösungsschritte entwickelt werden und
  - Daten als Lösungselemente definiert werden.

Das vollzieht sich in verschiedenen Abstraktionsstufen, d. h. Schichten, unter Nutzung der verschiedenen Kerne.

Eine solche Betrachtung findet man in [4, 8] nicht, dort wird verfahrensorientiert vorgegangen und die allgemeine Struktur und ihr Einfluss auf das Vorgehen treten zurück.

4 Nutzungstechnologie

Unter dieser Überschrift wurde die Frage behandelt, **wie** die zu bearbeitende Aufgabe mittels Programmsystem im Zusammenspiel mit dem Nutzer gelöst wird. Durch seine Arbeit am Programmsystem INTKOS ([2], Abschnitt 7.2) hatte Kuno Schmidt hier insbesondere seine Erfahrungen einbringen können.

Wieder wurde darauf hingewiesen, dass die Kommunikation zwischen Nutzer und Programmsystem über eine geeignete, nutzergerechte (Fach)Sprache erfolgen sollte. In der Neuzeit tritt an diese Stelle die grafische Nutzerschnittstelle (Oberfläche), die man gutwillig auch als grafische Sprache ansehen kann.

Bei der Bearbeitung einer Aufgabe, global oder lokal als einzelner Lösungsschritt, werden die drei Phasen

- Vorbereitung,
- Durchführung und
- Nachbereitung

betrachtet, die sich in ihren Spezifika durch die verschiedenen Nutzungsformen unterscheiden. Deren Unterscheidung wurde vorgenommen nach

- der Stellung des Nutzers zum Arbeitsmittel, d. h. ob er das Programmsystem **direkt** oder **indirekt** (über einen Betreiber) nutzt,
- der Art der Bearbeitung, d. h. ob die Lösung im **Dialog** mit dem Nutzer oder im **Monolog** (Stapelbetrieb, ohne Einflussnahme des Nutzers) erfolgt und
- dem Modus der „Trassenfindung“, d. h. ob der Nutzer **aktiv** Einfluss auf den Lösungsprozess nimmt oder ob er bis auf die Aufgabenstellung die Lösung automatisch bestimmen lässt, also **passiv** bleibt.

Anstelle des „Nutzers“ ist in [4] und [8] der Begriff des „Benutzers“ getreten, allerdings in der einschränkenden Bedeutung eines direkten Nutzers in unserem Sinne ([4], Seite 24):

> “**Benutzer** sind nur diejenigen Personen, die ein Computersystem unmittelbar einsetzen und *bedienen*, oft auch Endnutzer oder Endanwender genannt.“

Im Rahmen der Software-Ergonomie werden dann detailliertere Untersuchungen zum möglichen Benutzerverhalten (Anfänger, Gelegenheitsbenutzer, Ex-

perte) gemacht und daraus Gestaltungs- und Bewertungskriterien (etwa für die grafische Benutzeroberfläche) abgeleitet.

In [8], Seite 55, wird zwischen aktiven und passiven Benutzern hinsichtlich ihres Verhaltens bei der Informationssuche unterschieden, was unserem Standpunkt nahe kommt.

Meinen beruflichen Erfahrungen der letzten Jahre an der Universität sagen allerdings aus, dass es immer noch auch unter den ProfessorInnen indirekte Nutzer gibt, die sich scheuen oder weigern, direkt mit dem Computer in Nutzung zu treten. Das sollte weiterhin nicht unberücksichtigt bleiben.

Neben diesen Betrachtungen wurden auch Hinweise zur Wartung und Modifizierung gemacht. Dabei waren die angesprochenen Methoden auf die damals vorhandenen Programmiertechniken bezogen. CASE-Werkzeuge standen, jedenfalls in Verfügung der Interessengruppe, nicht bereit.

5 Architektur

Als Begriffsbestimmung wurde in [2] formuliert:

> „Die Architektur eines Programmsystems umfasst die funktionelle und programmtechnische Struktur sowohl des Gesamtsystems als auch seiner Elemente und Teilsysteme. Sie beschreibt die Strukturkomponenten, die Relationen, in denen sie untereinander in Beziehung stehen, sowie die Voraussetzungen, unter denen sie effektiv erzeugt und genutzt werden können."

Zu diesem Standpunkt hat sich kaum etwas geändert. Balzert schreibt ([4], Seite 639):

> „Eine Softwarearchitektur beschreibt die Struktur des Softwaresystems durch Strukturkomponenten und ihre Beziehungen untereinander."

Es fehlen hier nur die Bemerkungen zu den Voraussetzungen, also die Einbettung in die Umgebung, die uns wichtig erschienen.

Ausführlich wurde auf das Modulkonzept für die Bildung der Strukturkomponenten der Architektur eingegangen. Die strukturelle bzw. modulare Programmierung befand ja in der damaligen Zeit in großer Blüte. Als Begriffsbestimmung wurde formuliert:

> „Ein Modul ist eine Zusammenfassung von Ressourcen. In seiner Schnittstelle zur Umwelt (dem Interface) wird exakt festgelegt, welche Ressourcen in welcher Form er (nach außen) zur Verfügung stellt, und welche er (von außen) benötigt."

Interessant ist die vorgenommene Unterscheidung zwischen *Beschreibung*, *Generierung* und *Nutzung* von Moduln. Aus der Beschreibung, in der die Schnittstelle und der innere Aufbau eines Moduls festgelegt sind, entstehen durch Generierung reale Moduln. Zur gleichen Beschreibung unterschiedlich generierte Moduln können durchaus unterschiedliche Ressourcen bereitstellen.

Die Idee der Generierung hat sich gegenwärtig in der generischen Programmierung und deren Ausprägung bei generischen Programmiersprachen

(Templates, generisches C\#) manifestiert.

Eine zweite Unterscheidung betrifft die Einteilung in *Programmoduln*, die zur Bearbeitung von Daten dienen, also die Lösungsschritte realisieren (Problemmoduln) und organisieren (Steuermoduln, Dienstmoduln), und *Datenmoduln*, die Lösungselemente darstellen (Problemdaten) sowie Lösungszustände beschreiben (Organisationsdaten). Die Datenmoduln stellen auch Funktionen als Ressourcen zur Verfügung. Insofern entsprechen sie der Idee der Datenkapselung.

Zwischen den Moduln wurde als wesentliche Beziehung das gegenseitige *Aktivieren* betrachtet, wobei insbesondere auch die Nutzung von Ressourcen eines Moduls als Aktivieren verstanden wurde. Jegliche Möglichkeit des gegenseitigen Aktivierens sollte in der Schnittstelle detailliert, also auch die spezielle Erlaubnis der Nutzung gewisser Ressourcen durch gewisse Module, fixiert sein. Über die Aktivierungs-Beziehung wurden verschiedene Strukturierungsformen (Symmetrie, Rekursionsfreiheit, Hierarchie) hinsichtlich ihrer Vor- und Nachteile bzw. sonstigen Konsequenzen diskutiert.

Mit der Ära der Objektorientierung wurde das Modulkonzept abgelöst und leider abgeschwächt. An Stelle des Moduls trat das Paket, der Namensraum oder die Klasse. Zwar wurde der Begriff der Schnittstelle beibehalten, im obigen Sinne handelt es sich dabei aber mehr um einen Teil der Beschreibung einer Klasse. Einschränkungen der Nutzung durch andere Klassen (bzw. deren Instanzen) sind nur in genereller Form (alle oder keine über öffentlich oder privat) und bezüglich der Vererbungshierarchie (geschützt) möglich. Pakete und Namensräume fassen lediglich Klassen zusammen, die nach außen freigegeben oder gesperrt werden können.

6 Steuerung

Die konsequente Auslagerung einer separaten Steuerung geht auf das Konzept zurück, dass Lösungsvorschriften als Szenarien in der Szenenbasis abgelegt sind und durch eine Steuerung zu interpretieren oder zu kompilieren sind. Die explizite Trennung der Steuerung von den einzelnen Lösungsschritten bringt mehr Übersicht, bessere Chancen für die Verifikation und eine leichtere Wartung bzw. Modifikation.

Die im Buch diskutierten unterschiedlichen Formen von Szenarien wie Zustandsautomaten, ablauforientierte oder datenflussorientierte Szenarien findet man in der UML (Unified Modeling Language) bei den verschiedenen Diagrammen wieder [5, 6]. Der Unterschied besteht aber darin, dass die UML ein Modell beschreibt, auf dessen Grundlage dann eine Implementierung nach dem Geschmack des Programmierers entsteht, während die Szenenbasis die Implementierung bereits einschließt.

Mit der fast ausschließlichen Kopplung einer Anwendung an eine grafische Nutzeroberfläche ist die Ereignissteuerung der Anwendung unvermeidbare Konsequenz. Dies hat sicher gewisse Vorteile, aber auch den Nachteil, dass in der Implementierung schnell die Übersicht über mögliche Ereignisfolgen und deren Auswirkungen verloren geht. In [3] wurde ein Konzept entwickelt, wie man aus einer separaten Spezifikation der Ereignisse und ihrer Zusammenhänge automatisch eine Implementierung gewinnen kann und zugleich Mechanismen zur Verifikation bereitstellt.

Auch die objektorientierte Programmierung verteilt die Steuerung auf die gesamte Anwendung. Das Grundkonzept eines Objekts als Datenkapsel, die auch die Zugriffsroutinen einschließt, ist grundsätzlich zu begrüßen. Das war bei den Datenmoduln analog vorgesehen. Ein problematischer Nebeneffekt in der Objektorientierung (siehe zum Beispiel SMALLTALK) besteht aber darin, dass Objekte über die Methoden Informationen austauschen sollen und dadurch im Wesentlichen der gesamte Programmablauf definiert ist. Falls man diesem Konzept konsequent folgt, wird dadurch die Steuerung integriert und nicht separiert.

7 Entwicklungstechnologie

Die Entwicklungstechnologie war kein Schwerpunkt der Untersuchungen der Interessengruppe. Es wurden deshalb nur allgemeine Betrachtungen aufgenommen, die entweder aus der Literatur entnommen wurden oder sich aus den eigenen Erfahrungen ergaben. Es wurden lediglich die drei Phasen

- Planung,
- Entwurf und
- Implementierung

unterschieden.

Diese sehr allgemein gehaltene Betrachtungsweise wird mittlerweile viel detaillierter behandelt. Balzert [4] unterscheidet

- Planung,
- Definition,
- Entwurf,

- Implementierung,
- Abnahme und Einführung sowie
- Wartung und Pflege.

Die jetzt intensivere und genauere Beschäftigung mit der Entwicklungstechnologie ist der Tatsache geschuldet, dass Softwareentwicklung von einer individuellen Tätigkeit zunehmend zu einer Produktionsform übergegangen ist. Die Mitglieder der Interessengruppe waren damals nur in kleinen Kollektiven oder gar als Einzelperson in der Softwareentwicklung tätig.

8 Modellierung

Die Modellierung wurde einschränkend als *mathematische* Modellierung verstanden. Drei Ziele sollten verfolgt werden:

1. Das formale Modell soll die Grundlage einer konzeptuellen Ebene sein, einschließlich der Angabe von Transformationsregeln zur externen Ebene (Schnittstelle zum Nutzer) und zur internen Ebene (Implementierung).
2. Aus dem Modell sollen praxisrelevante Aussagen wie Struktureigenschaften, Zuverlässigkeit, Datenflusseigenschaften, Effizienz, usw. gewonnen werden.
3. Die Modelle sollen eine generelle Entwurfsgrundlage für Programmsysteme liefern.

Die Arbeit der Interessengruppe war auf das zweite Ziel konzentriert, unterschieden nach determinierten und stochastischen Methoden.

Bei ersterem wurde die Datenflussanalyse in sogenannten **schwach interpretierten Programmschemata** nach [1] genauer vorgestellt. In einem Programmschema wird die Bedeutung der Operatoren (Lösungselemente) ignoriert, sie wird erst durch eine Interpretation hinzugefügt. Schwache Interpretation bedeutet, dass lediglich gewisse Vorgaben bzw. Bedingungen an die Bedeutungen, wie zum Beispiel Invarianz bezüglich Werteänderung gewisser Datenbereiche, gemacht werden. Durch die Datenflussanalyse kann man zum Beispiel Hinweise auf die strukturelle Gestaltung und auf die Steuerung gewinnen. Das kann zur Optimierung des Systems genutzt werden. Diese Technik wurde ursprünglich für die Programmoptimierung in Compilern entwickelt, kann aber auf Programmsysteme übertragen werden, falls die Steuerung als separate Komponente existiert.

Bei den stochastischen Methoden wurden Markov-Ketten eingesetzt, um das Leistungsverhalten zu analysieren. Zum Beispiel kann man unter gewissen Voraussetzungen ermitteln, mit welcher Wahrscheinlichkeit welche Zustände erreicht werden. Man hat damit ein gewisses Maß für die Zuverlässigkeit eines Programmsystems, wenn man sich etwa auf gewünschte Endzustände oder

unerwünschte Fehlerzustände bezieht. Auch Überschlagsrechnungen zur erwarteten Rechenzeit sind möglich.

Es wird nochmals betont, dass diese Betrachtungen als Grundvoraussetzung eine entsprechende Architektur des Programmsystems haben!

Im Rahmen der Softwaretechnik wird intensiv mit Modellen gearbeitet, UML soll dazu die einheitliche Sprache sein. Nach meinem Verständnis handelt es sich dabei aber nicht um mathematische Modelle, was eine mathematische Analyse unmöglich macht.

Zwar widmet sich in [8] der Teil 1 mit 136 Seiten der Systemanalyse, allerdings in einer sehr allgemeinen, aus meiner Sicht zum Teil sehr pragmatischen, Weise. Das kann ja methodisch nützlich sein, liefert aber keine objektiven und quantitativen Aussagen. Modellierung widmet sich hauptsächlich dem ersten Ziel.

Auch in [4] wird ein solcher Ansatz verfolgt, Objektorientierung (Analyse, Entwurf, Implementierung) als Methode der 90er Jahre proklamiert. Es wird ausgesagt, dass bis 1994 etwa 50 verschiedene Analysemethoden entwickelt wurden, eine Wertung habe ich nicht festgestellt.

Natürlich existieren weiterhin umfangreiche Untersuchungen zur mathematischen Modellierung von Software (Programmsystemen). Dazu gehören zum Beispiel modale und temporale Logik, Model-Checking, Petri-Netze, algebraische Ansätze usw. In die Software-Technik echt integriert scheinen diese aber nicht zu sein.

9 Resümee

Die Interessengruppe hatte sich bereits zur damaligen Zeit einem Gegenstand gewidmet, der heute sehr aktuell ist, sich natürlich in der Betrachtungsweise, Schwerpunktsetzung und Begriffsbildung gewandelt hat. Die verfügbare Rechentechnik (Hardware und Software) war damals wesentlich bescheidener bezüglich der zeitlichen und räumlichen Ressourcen und der möglichen Nutzungsformen. Die Software*produktion* stand am Anfang und wurde von den Mitgliedern der Interessengruppe in ihrer Entwicklung nicht prognostiziert. Das hatte Konsequenzen auf die gezogenen Schlussfolgerungen.

Als bemerkenswerte Gedanken möchte ich zusammenfassend herausstellen:

- die Einordnung von Programmsystemen in allgemeine Aufgabenlösungsprozesse mit der Strukturierung des Arbeitsmittels in Rahmen und Basen,
- die Orientierung auf die Nutzung entsprechender Sprachen für die Beschreibung der verschiedenen Entwicklungsschritte und Komponenten,
- die Betrachtungen zur Nutzungstechnologie,

- die Unterscheidung verschiedener Modularten im Rahmen der Architektur,
- die separate Behandlung der Steuerung eines Programmsystems und
- die konkreten Methoden zur Modellierung.

Leider wurde das Buch nur in einem gewissen Teil der Welt zur Kenntnis genommen (1988 im Moskauer Verlag Mir auf Russisch herausgegeben). Es wird in der modernen Literatur zur Software-Technik nicht zitiert, also hatte es kaum oder keinen Einfluss auf die entsprechende Entwicklung, die in einigen Positionen andere Wege ging. Ob diese anderen Wege in jeder Hinsicht besser sind und sich bewähren werden, ist aber noch nicht entschieden.

10 Literatur

[1] BACHMANN, P. (1977): Data-Flow-Analysis in Weakly Interpreted Program-Schemes. *Information Processing 77, North-Holland Publishing Company (1977), S. 63-68*.

[2] BACHMANN, P. (Hrsg.) (1983): Programmsysteme: Anwendung-Entwicklung-Fundierung. *Akademie-Verlag Berlin.*

[3] BACHMANN, P. (2006): Formal Verification of Event Driven Systems. *BTU-Cottbus, Computer Science Reports 03/06, October 2006, S. 43-54*.

[4] BALZERT, H. (1996): Lehrbuch der Softwaretechnik. *Heidelberg, Berlin, Oxford: Spektrum Akademischer Verlag.*.

[5] BURKHARDT, R. (1997): UML – Unified Modeling Language. *ADDISON-WESLEY Longman Verlag GmbH*.

[6] FOWLER, M. & SCOTT, K. (1998): UML konzentriert. *ADDISON-WESLEY Longman Verlag GmbH*.

[7] HANTZSCHMANN, K. (2008): N. J. Lehmann – sein Wirken für die Informatik in der DDR. *3. Symposium „Informatik in der DDR", Dresden.*

[8] KOREIMANN, D. S. (2000): Grundlagen der Software-Entwicklung. *München, Wien: R.Oldenbourg Verlag*.

[9] LEHMANN, N. J. (1974): Aufbau und Organisation von Programmpaketen. *TU Dresden, Schriftenreihe WBZ MKR/IV, Heft 8/74, S. 1-7*.