Versions-, Schnittstellen- und Konfigurationskontrolle

Hans-Peter Subel
Werum Datenverarbeitungssysteme GmbH
Glogauer Straße 2 A
2120 Lüneburg

Zusammenfassung :

In diesem Artikel wird das Software-Entwicklungswerkzeug VICO (Version, Interface and Configuration Control) vorgestellt. VICO wird eingesetzt in der Feinspezifikations-, Programmierungs-, Test- und Wartungsphase eines Projekts und unterstützt sowohl die Entwicklung als auch die Verwaltung komplexer modularer Programmsysteme.

VICO legt sämtliche Dokumente eines Projekts (Quelltexte, Dokumentationen, Objektcodedateien, ...) zusammen mit Verwaltungsinformationen (Erstellungsdaten, Zustandsangaben, ...) und Informationen über Beziehungen zwischen Dokumenten (ist Objektcode von, wird benutzt in, ...) in einer Datenbank ab. Auf der Basis dieser Informationen verwaltet VICO die verschiedenen Versionen eines Dokumentes, überwacht die Veränderungen der Schnittstelle eines Moduls und ermöglicht eine sichere Konfigurierung von Programmen.

1. Einleitung

VICO (Version, Interface and Configuration Control) ist entstanden aufgrund von Erfahrungen, die im Laufe mehrerer Jahre bei der Entwicklung modularer Programmsysteme in höheren Programmiersprachen (PEARL, C, PASCAL, PL/I, Ada, ...) gesammelt worden sind. Es wurde festgestellt, daß besonders in der Feinspezifikations-, Programmierungs-, Test- und Wartungsphase eines Projekts viele kostenintensive, aber durchaus vermeidbare Probleme aufgetreten sind, die zumeist einer der folgenden Problemklassen zugeordnet werden können:

- Schnittstellenprobleme
- Versionsprobleme
- Konfigurationsprobleme

Diese Problemklassen sollen anhand von Fragestellungen erläutert werden, denen sich der Programmierer in der täglichen Praxis ständig ausgesetzt sieht.

Schnittstellenprobleme

- Welche Moduln müssen nach einer Änderung der Schnittstelle eines Moduls ebenfalls angepaßt und neu übersetzt werden ?
- In welchen Moduln wird ein bestimmtes globales Objekt benutzt ?

Versionsprobleme

- Welche Versionen eines Moduls gibt es ?
- Wer hat die Version implementiert ?
- Worin unterscheiden sich die Versionen ?
- In welchen (Programm-) Konfigurationen wird eine bestimmte Version benutzt ?
- Mit welchem Compiler und mit welchen Optionen wurde die Version übersetzt ?
- Kann die Version gelöscht werden ?

Konfigurationsprobleme

- Aus welchen Moduln besteht eine bestimmte (Programm-) Konfiguration ?
- Ist die Konfiguration aktuell (müssen einige Moduln noch übersetzt werden, oder genügt es, neu zu binden) ?
- Auf welchen Rechnern, bei welchen Kunden ist die Konfiguration installiert ?
- Wo ist das aktuelle Benutzerhandbuch einer bestimmten Konfiguration zu finden ?

Zur Beantwortung dieser Fragestellungen und zur Lösung der damit verbundenen Probleme haben wir, aufbauend auf Ideen (vgl. /1/, /2/, /3/), die im Umfeld neuerer Programmiersprachen (Ada /4/, MODULA-2 /5/, ..) und nicht mehr ganz so neuer Betriebssysteme (UNIX /6/, /7/) entstanden sind, das weitgehend rechner- und betriebssystemunabhängige Werkzeug VICO zur Versions-, Schnittstellen- und Konfigurationskontrolle entwickelt.

2. Die VICO-Datenbank

Die grundlegende Idee bei der Entwicklung von VICO bestand darin, sämtliche Informationen, die ein Projekt betreffen, konsequent in Relationen einer Datenbank (in diesem Fall BAPAS-DB /8/) abzulegen und die von der Datenbank bereitgestellten Mechanismen (Zugriffsschutz, Datensicherheit, parallele Aktivitäten auf einem Datenbestand, Transaktionen, Anfragesprache, ...) bei der Implementierung von VICO intensiv zu nutzen.

Bild 1 beschreibt die Konventionen, die in den nachfolgenden Beispielen für die Darstellung von Datenbank-Relationen verwendet werden.

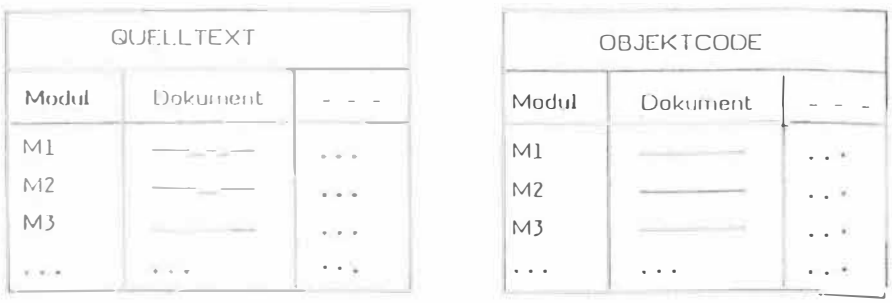
RELATION		
Schlüsselattribut	Attribut	- - -
Wert	Wert	. . .
. . .	. . .	. . .

RELATION	:	Name einer Relation
Schlüsselattribut	:	Name eines Schlüsselattributs
Attribut	:	Name eines Attributs
- - -	:	Weitere, namentlich nicht genannte Attribute
Wert	:	Wert eines Attributs
. . .	:	Beliebiger, nicht angegebener Wert eines Attributs

Bild 1 : Konventionen für die Darstellung von Relationen.

Zunächst werden alle Dokumente eines Projekts in Relationen der VICO-Datenbank eingetragen. Dazu gehören sowohl manuell mit Hilfe von Editoren erstellte Dokumente (Quelltexte, Spezifikationen, Handbücher, Graphiken, ...) als auch automatisch mit Hilfe von Programmen erstellte und damit reproduzierbare Dokumente (Objektcodedateien, Fehlermeldungen, Diagramme, ...). Damit kann der Zugriff auf sämtliche Dokumente eines Projekts von VICO überwacht werden.

Bild 2 zeigt Beispiele für Relationen, die ein Dokument enthalten. Bei dem Dokument handelt es sich um den Quelltext bzw. den Objektcode eines Moduls. Die Werte eines Dokuments, in diesem Fall der Quelltext und der Objektcode eines Moduls, werden von VICO verwaltet, aber in keiner Weise interpretiert oder bearbeitet.



(Werte eines Dokuments werden durch einen Strich symbolisiert.)

Bild 2 : Die Relationen QUELLTEXT und OBJEKTCODE.

VICO sind die Beziehungen zwischen den Dokumenten verschiedener Relationen implizit bekannt. Die Zuordnung erfolgt über identische Schlüsselwerte.

Beispiel für eine implizit bekannte Beziehung:

Dokumente der Relation OBJECTCODE werden aus Dokumenten der Relation QUELLTEXT generiert.

Beispiel für eine Zuordnung:

Der Quelltext und der daraus generierte Objektcode des Moduls "M1" sind in den Relationen QUELLTEXT und OBJECTCODE unter dem gleichen Schlüsselwert, nämlich "M1", abgelegt.

Diese Relationen werden um weitere Verwaltungsinformationen ergänzt:

- Zeitangaben
  (Erstellungs-, Zugriffs-, Übersetzungsdatum, ...)
- Benutzerangaben
  (Benutzerkennung, Zugriffsrechte, ...)
- Statusangaben
  (der Objektcode ist aktuell, der Programmcode ist aktuell)
- Entstehungsparameter
  (benutzter Compiler, verwendete Optionen, ...)

Bild 3 zeigt einen Ausschnitt aus der Relation OBJEKTCODE.

OBJEKTCODE					
Modul	übersetzt	gebunden	aktuell	Dokument	- - -
M1	29.01.86	30.01.86	ja	———	. . .
M2	23.01.86	30.01.86	ja	———	. . .
M3	10.01.86	29.01.86	nein	———	. . .
. . .	. . .	. . .	. . .	. . .	. . .

Bild 3 : Die Relation OBJEKTCODE.

Die Verwaltungsinformationen werden von VICO laufend aktualisiert.

In weiteren Relationen der Datenbank werden Informationen über die Beziehungen zwischen verschiedener Relationen vom VICO-Benutzer explizit eingetragen.

Bild 4 zeigt als Beispiel die Relation KONFIGURATION-MODUL, die beschreibt, welche Module zu einer (Programm-) Konfiguration gehören.

KONFIGURATION-MODUL		
Konfiguration	Modul	- - -
K1	M1	. . .
K1	M2	. . .
K1	M3	. . .
K2	M1	. . .
K2	M2	. . .
K2	M4	. . .
. . .	. . .	. . .

Bild 4 : Die Relation KONFIGURATION-MODUL.

Die implizit bekannten sowie die vom Benutzer explizit bekanntgemachten Informationen über die Beziehungen zwischen den Relationen sind für VICO von besonderer Bedeutung, da genau sie es ermöglichen, Regeln zu formulieren, nach denen im Falle der Änderung eines Dokuments automatisch zu verfahren ist.

Beispiel für eine Regel, nach der VICO verfährt:

> Nach der Änderung des Quelltextes eines Moduls ist der Objektcode des Moduls neu zu generieren. Anschließend sind alle Konfigurationen, in denen dieser Modul benutzt wird, neu zu binden. Der Benutzer des Moduls ist darüber zu informieren, daß die Dokumentation des Moduls nicht mehr aktuell ist.

Im folgenden wird exemplarisch anhand einiger Beispiele erläutert, wie VICO auf der Basis der in der VICO-Datenbank abgelegten Informationen die Schnittstellen-, Versions- und Konfigurationskontrolle realisiert.

3. Schnittstellenkontrolle

Die Moduln einer (Programm-) Konfiguration stehen über im- und exportierte Objekte in einer beliebig komplexen Beziehung. Ein Objekt in diesem Sinne ist eine Prozedur, eine Variable, ein Typ, eine Konstante oder ein entsprechendes, von der in einem Projekt verwandten Programmiersprache bereitgestelltes Konstrukt.

In VICO werden die Definitionen dieser Objekte grundsätzlich vom übrigen Quelltext des Moduls getrennt (vgl. /9/) und einzeln in Tupeln der Relation OBJEKT abgelegt (Bild 5).

OBJEKT		
Objekt	Definition	- - -
O1	DCL O1 FIXED (15) ;	...
O2	DCL TYPE O2 STRUCT [...] ;	...
O3	DCL O3: PROCEDURE (...) ;	...
...	...	...

Bild 5 : Die Relation OBJEKT.

Für jedes Objekt existiert damit genau eine Definition in der gesamten Datenbank, so daß Inkonsistenzen ausgeschlossen sind. Die Informationen über die Benutzung dieser Objekte werden in der Relation MODUL-OBJEKT eingetragen.
In dem Beispiel aus Bild 6 exportiert der Modul "M1" die Objekte "O1" und "O2". Modul "M2" importiert "O1" und "O2" von M1 und exportiert seinerseits "O3".

MODUL-OBJEKT			
Modul	Exporteur	Objekt	- - -
M1	M1	O1	. . .
M1	M1	O2	. . .
M2	M1	O1	. . .
M2	M1	O2	. . .
M2	M2	O3	. . .
. . .	. . .	. . .	. . .

Bild 6 : Die Relation MODUL-OBJEKT.

Diese beiden Relationen können bereits in der Feinspezifikationsphase eines Projekts gefüllt und als Spezifikation der Schnittstelle eines noch zu implementierenden Moduls verwandt werden.
Da der Zugriff auf die Objektdefinitionen von VICO kontrolliert wird, können alle von der Änderung betroffenen Module automatisch festgestellt und Übersetzungen eingeleitet werden, indem die von einem Modul jeweils benötigten Objektdefinitionen der Datenbank entnommen und mit dem übrigen Quelltext des Moduls zu einer Übersetzungseinheit zusammengefügt werden.

Der Aufwand für die Generierung dieser Übersetzungseinheit hängt von der in einem Projekt eingesetzten Programmiersprache ab. Der Aufwand ist gering, wenn die Sprache bereits eine Trennung von Schnittstellendefinition und Implementierung vorsieht (DEFINITION MODULE und IMPLEMENTATION MODULE in MODULA-2 oder PACKAGE-Konzept in Ada) und entsprechend höher, falls die Sprache eine Trennung nicht unterstützt (PEARL, C, PASCAL, PL/1, ...). In diesem Fall wird die Trennung mit Hilfe eines Preprozessors nachgebildet.

4. Versionskontrolle

Der VICO-Benutzer hat die Möglichkeit, verschiedene Versionen eines (Relations-) Tupels anzulegen. Diese Versionen können manuell (z.B. mit einem Editor) oder automatisch (z.B. mit einem Compiler) aufgrund der Wahl unterschiedlicher Optionen bzw. Parametereinstellungen (z.B. Übersetzung mit oder ohne Codeoptimierung) erzeugt werden.

Bild 7 zeigt ein Beispiel für verschiedene manuell erstellte Versionen des Quelltextes eines Moduls. Vom Quelltext des Moduls "M1" gibt es zwei Versionen, während vom Quelltext des Moduls "M2" zunächst lediglich eine Version existiert. Aus dem Quelltext des Moduls "M1", Version "1" sind aufgrund der verschiedenen Compileroptionen "option1" bzw. "option2" automatisch zwei Objektcode-Versionen generiert worden.

QUELLTEXT

Modul	Version	- - -
M1	1	...
M1	2	...
M2	1	...
...	...	...
...	...	...

OBJEKTCODE

Modul	Version	Option	- - -
M1	1	option1	...
M1	1	option2	...
M1	2	option1	...
M2	1	option3	...
...	...	...	...

Bild 7 : Die um eine Versionsnummer erweiterten Relationen QUELLTEXT und OBJEKTCODE.

VICO verwaltet die verschiedenen Versionen eines Tupels und überwacht deren Verwendung. Nach dem Löschen oder Ändern eines Quelltextes werden beispielsweise alle aus dieser Version automatisch generierten Dokumente (Objektcodedateien, ausführbare Programme, ...) ebenfalls gelöscht oder neu generiert.

Da die bei der Generierung benutzten Optionen in der Datenbank abgelegt werden, kann der VICO-Benutzer zu einem späteren Zeitpunkt auch ohne Übersetzungsprotokoll feststellen, aus welcher Version eines Quelltextes, unter Verwendung welcher Optionen ein bestimmter Objektcode entstanden ist.

5. Konfigurationskontrolle

Mit Hilfe der Relation KONFIGURATION-MODUL (Bild 8) legt der VICO-Benutzer fest, welche Versionen eines Moduls zu einer bestimmten Konfiguration gehören.

KONFIGURATION-MODUL				
Konfiguration	Modul	Version	Option	- - -
K1	M1	1	option1	. . .
K1	M2	3	option2	. . .
K1	M3	1	option1	. . .
K2	M1	2	option1	. . .
K2	M2	1	option3	. . .
K2	M4	1	option2	. . .
. . .	. . .	. . .	. . .	. . .

Bild 8 : Die Relation KONFIGURATION-MODUL.

Diese Informationen wertet VICO aus, erzeugt einen Binde-Job und sorgt letztlich für die Generierung des gewünschten Programmcodes.

Da der VICO-Benutzer keine eigenen Binde-Jobs erstellen kann, handelt es sich bei dieser Relation um die einzige und exakte Aufstellung der zu einer Konfiguration gehörenden Moduln.
Der generierte Programmcode wird als Dokument in die Relation PROGRAMMCODE eingetragen (Bild 9).

PROGRAMMCODE			
Konfiguration	aktuell	Dokument	- - -
K1	ja	———	. . .
K2	nein	———	. . .
. . .	. . .	. . .	. . .

Bild 9 : Die Relation PROGRAMMCODE.

Änderungen des Quelltextes eines in der Konfiguration benutzten Moduls führen dazu, daß der Programmcode dieser Konfiguration als nicht mehr aktuell gekennzeichnet wird.

6. Die VICO-Benutzerschnittstelle

Der Benutzer kommuniziert mit VICO über eine wahlweise menü- oder kommandogesteuerte Dialogschnittstelle.
Prinzipieller Befehlsaufbau:

- Angabe der Funktion
- Angabe der Relation
- Anwahl von Tupeln der Relation.

Die Anwahl der Tupel einer Relation erfolgt mit Hilfe eines vereinfachten Query-by-example-Mechanismus /10/, /11/. Auf dem Bildschirm erscheint die auf Schlüsselattribute reduzierte Schablone eines Tupels der gewünschten Relation. Alle Spalten sind mit einem Fragezeichen vorbelegt, das einen beliebigen Schlüsselwert symbolisiert. Der Benutzer hat nun die Möglichkeit, die Werte dieser Schlüsselattribute festzulegen (Bild 10).

Beispiel für die Anwahl aller Konfigurationen, die die Version "2" des Moduls "M1" benutzen:

KONFIGURATION-MODUL			
Konfiguration	Modul	Version	Option
?	M1	2	?

Beispiel für die Anwahl des Quelltextes der Version "2" des Moduls "M1" :

QUELLTEXT	
Modul	Version
M1	2

Bild 10 : Anwahl von Tupeln einer Relation.

Die wichtigsten Funktionen von VICO sollen in diesem Rahmen nur kurz vorgestellt werden.

LIST

Die LIST-Funktion dient zur Beantwortung von Standardanfragen an die VICO-Datenbank. Zusammen mit dem Query-by-example-Mechanismus stellt sie die Basis eines komfortablen und umfassenden Auskunftssystems dar. Für weitere spezielle Anfragen steht dem Benutzer die Anfragesprache des Datenbanksystems BAPAS-DB zur Verfügung.

INSERT

Die INSERT-Funktion wird benutzt, um Tupel in die Relationen der Datenbank einzutragen.

EDIT

Die EDIT-Funktion ermöglicht die textuelle Bearbeitung von Dokumenten, die in den Relationen der Datenbank abgelegt sind (Quelltexte, Objektdefinitionen, Handbücher, ...). Nach Beendigung der Bearbeitung eines Dokuments werden die Zustandsinformationen sämtlicher von dieser Änderung betroffenen Relationstupel von VICO automatisch aktualisiert.

COMPILE

Mit Hilfe der COMPILE-Funktion werden automatisch alle nicht mehr aktuellen Objektcodedateien der in einer Konfiguration benutzten Moduln neu generiert und gebunden. Eine Objektcodedatei ist nicht mehr aktuell, wenn der Quelltext des Moduls oder die Definition eines im- oder exportierten Objekts geändert wurde. Nach Abschluß dieser Transaktion werden die Zustandsinformationen der betroffenen Tupel aktualisiert und Fehlermeldungen sowie Protokolle in die entsprechenden Relationen eingetragen.
Diese Funktion ersetzt vollständig die sonst üblichen, vom Programmierer zu initiierenden und anschließend zu kontrollierenden Übersetzungs- und Bindevorgänge.

PUT, GET

Mit Hilfe der GET-Funktion können Dokumente aus Relationen der VICO-Datenbank in Dateien des vom VICO-Anwender benutzten Betriebssystems kopiert werden.
Die PUT-Funktion ermöglicht die Übernahme von manuell erstellten Dokumenten in die Relationen der VICO-Datenbank.

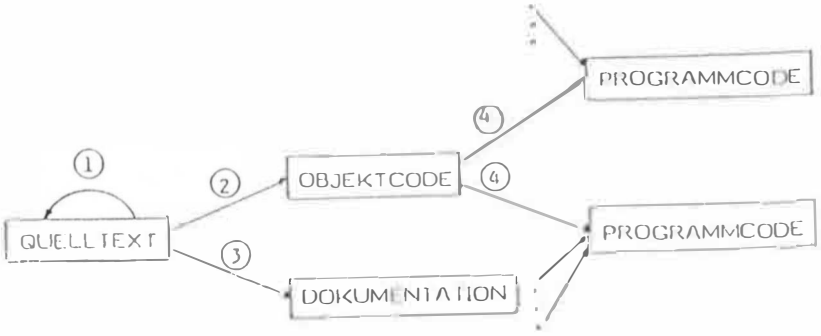
Weitere Funktionen regeln den Mehrbenutzerbetrieb und den parallelen Zugriff auf gleiche Datenbestände.

7. Datenbankmechanismen für Programmierumgebungen

VICO ist als Datenbankanwendungsprogramm implementiert worden. Zur Zeit wird überlegt, welche Komponenten von VICO als allgemein verfügbare Datenbankmechanismen realisiert werden können (vgl. /12/).

Event / Trigger - Mechanismen

Eine Hauptaufgabe von VICO besteht darin, nach der Modifikation eines Tupels einer Relation eine Vielzahl relationsabhängiger Aktionen durchzuführen, die ihrerseits Modifikationen in Tupeln weiterer Relationen bedingen, usw. Ein Beispiel hierzu ist in Bild 11 zu finden.



Aktion	(1)	:	Ändern des Quelltextes eines Moduls
Aktion	(2)	:	Übersetzen des Quelltextes und Generierung des Objektcodes eines Moduls
Aktion	(3)	:	Aktualisieren der Dokumentation eines Moduls
Aktion	(4)	:	Binden des Programmcodes sämtlicher Konfigurationen, die den geänderten Objektcode benutzen

Bild 11 : Auswirkungen der Änderung eines Quelltextes.

Es liegt nahe, bereits zum Zeitpunkt der Definition einer Relation (DDL-Sitzung) anzugeben, welche Aktionen nach der Modifikation (Einfügen, Ändern, Löschen) eines Tupels dieser Relation durchzuführen sind. Diese Aktionen sind vom Datenbankanwender zu programmieren. Den Zeitpunkt ihrer Aktivierung bestimmt das Datenbanksystem zur Laufzeit der Datenbankanwendung.

Dieser sogenannte event / trigger - Mechanismus (/13/) unterstützt und vereinfacht die Realisierung von Programmierumgebungen, so daß daran gedacht werden kann, die Programmierung der Aktionen dem späteren Benutzer selbst zu überlassen, um ihm die Möglichkeit zu geben, auf der Basis bereits vorhandener Standardkomponenten von VICO, eine den speziellen Bedürfnissen des Anwendungsbereichs angepaßte Programmierumgebung generieren zu können.

Archivierungseinheiten

Eine weitere Aufgabe von Programmierumgebungen, die zur Zeit von VICO noch nicht adäquat gelöst wird, resultiert aus der Notwendigkeit, logisch zusammenhängende Teile von Informationen eines Projekts gemeinsam zu archivieren. Diese Anforderung entsteht beispielsweise dann, wenn der Programmcode einer bestimmten Konfiguration zur Benutzung freigegeben wird, da anschließend noch sämtliche Quellen für eventuell nachträglich notwendige Änderungen zur Verfügung stehen müssen (Einfrieren der Quellen).
Es ist nicht nötig und zumeist auch nicht gewünscht, immer sämtliche Informationen eines Projekts vollständig zu archivieren. So muß es u.a. auch möglich sein, die Archivierung auf Informationen über bestimmte ausgewählte Moduln eines Projekts zu beschränken.

Auch in diesem Fall liegt es nahe, bereits zum Zeitpunkt der Definition der Relationen (DDL-Sitzung) festzulegen, welche Relationen zu einer gemeinsamen Archivierungseinheit gehören und mit Hilfe welcher Kriterien die zu archivierende Information beschränkt werden kann. Das Datenbankanwendungsprogramm bestimmt zur Laufzeit, wann und in welchem Umfang die Archivierung tatsächlich durchzuführen ist.

Dieser Archivierungsmechanismus unterstützt ebenfalls die bereits erwähnte Absicht, die Generierung von Programmierumgebungen dem späteren Anwender selbst zu überlassen.

8. Implementierung und Einsatz

VICO ist in einer Sprache programmiert, die wahlweise automatisch auf PEARL, PL/I oder C abgebildet werden kann. VICO steht auf CADMUS 9200 unter MUNIX zur Verfügung und unterstützt zur Zeit eine Programmentwicklung in PEARL, C, PL/I und PASCAL.

Erstmalig eingesetzt wurde VICO bei der Implementierung eines Compilers für eine umfangreiche Spezialsprache zur anwendungsorientierten Formulierung statistischer Problemlösungsverfahren. Das Programmsystem besteht aus 70 Moduln und ca. 1000 globalen Objekten. An der Realisierung waren Gruppen verschiedener Institutionen beteiligt.

Zur Zeit wird VICO bei der Implementierung eines Testsystems sowie bei der Realisierung eines verteilten Programmsystems zur Disposition und Schaltung eines komplexen Leitungsnetzes eingesetzt. Diese Projekte werden in PEARL realisiert.

Eine Weiterentwicklung von VICO, besonders im Hinblick auf die erwähnten Datenbankmechanismen, erfolgt im Rahmen des Verbundprojekts PROSYT (/14/).

Literatur

/1/ Habermann, A.N.: The Gandalf Research Project, Dept. of Comp. Sc. Research Review 1978 - 1979 (1980).

/2/ Tichy, W.T.: Software Development Control Based on System Structure Description, Ph. D. Thesis, Dept. of Comp. Sc., Carnegie-Mellon University (January 1980).

/3/ Cheatham, T.E.: Comparing Programming Support Environments, s. /4/ .

/4/ Habermann, A.N.; Perry, D.E.: System Composition and Version Control for Ada, Software Engineering Environments, Proceedings of the Symposium, June 16 - 20, 1980, Lahnstein, Hünke, H., North Holland Publishing Company.

/5/ Wirth, Niklaus: Programming in MODULA-2, Springer-Verlag, Berlin, Heidelberg, New York 1983.

/6/ Rochkind, M.J.: The Source Code Control System, IEEE Transactions on Software Engineering, SE - 1, 4 (Dec. 1975), 364 - 369.

/7/ Feldman, S.I.: MAKE - A Program for Maintaining Computer Programs, UNIX Programmer's Manual, Vol. 9, Apr. 1979, 255 - 265.

/8/ Goede, K.; Landwehr, K.: BAPAS-DB - Ein portables Datenbanksystem für Prozeßrechner, Informatik Fachberichte 39, Springer-Verlag Berlin, Heidelberg 1980, 443 - 452.

/9/ Warren, S.; Martin, B.C.; Hoch, C.: Experience with a Module Package in Developing Production Quality PASCAL Programs, Proceedings of the 6th International Conference on Software Engineering, 1982, Tokyo, IEEE Computer Society Press.

/10/ Zloot, M.M.: Query-by-Example: A data base language, IBM Systems Journal 16 (1977), 324 - 343.

/11/ Neves, J.C.: A Prolog Implementation of Query-by-Example, Proceedings of the International Computing Symposium, 1983, Nürnberg, Teubner.

/12/ Härder, T.; Reuter, A.: Architektur von Datenbanksystemen für Non-Standard-Anwendungen, in: Datenbank-Systeme für Büro, Technik und Wissenschaft, Informatik Fachberichte 94, Springer-Verlag Berlin, Heidelberg 1985, 253 - 286.

/13/ Dietrich, Klaus-R.; Kotz, Angelika; Mülle, Jutta: An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases, FZI-Publikationen 2, Forschungszentrum Informatik an der Universität Karlsruhe, Juni 1985.

/14/ Abbenhardt et al : Software-Engineering-Verbundprojekte, in: Computer Magazin, Oktober 1986, 67 - 78.