SOFTWARE-SPEZIFIKATION DURCH HALBFORMALE, ANSCHAULICHE MODELLE

Jochen Ludewig, Martin Glinz, Hans Matheis
Brown Boveri Forschungszentrum, CH-5405 Baden

Zusammenfassung

Der Beitrag erörtert die Frage, welche Ansätze zur Spezifikation grundsätzlich in Frage kommen, und begründet, warum in der industriellen Praxis das Prinzip der halbformalen Spezifikation auf der Basis anschaulicher Modelle vorteilhaft ist. Die Beispiele und die am Schluss wiedergegebenen Erfahrungen stammen aus unserer Arbeit mit dem Spezifikationssystem SPADES, das auf dem Prinzip der halbformalen Beschreibung beruht.

Einleitung

Spricht man von Spezifikationsproblemen, so können zwei ganz unterschiedliche Komplexe gemeint sein, nämlich die Schwierigkeit, eine Spezifikation zu erreichen, oder die, eine Spezifikation fehlerfrei in Code umzusetzen.

Natürlich ist beides nötig, um von einer vagen Idee zu einem Software-Produkt zu gelangen oder zu einem System, das Software enthält. Da wir aber heute keine Techniken kennen, mit deren Hilfe das Gesamtproblem systematisch bearbeitet werden kann, konzentrieren wir uns je nach Interessen, Erfahrungen und Prioritäten auf einen der beiden Aspekte. Nachfolgend werden die beiden entsprechenden Ansätze verglichen; der Einfachheit halber sind sie mit den Klischees "Mathematiker" und "Ingenieur" identifiziert.

Der Ansatz des Mathematikers erlaubt es nicht, fehlerhafte Programme zu akzeptieren. Darum stellt er die Verifikation des Programms oder besser noch die Programmerzeugung durch Semantik-erhaltende Transformationen in den Vordergrund. Beides bedarf einer streng formalen Spezifikation. So sind die Techniken der formalen Spezifikation entstanden (axiomatisch, denotationell, im Prädikatenkalkül usw.).

Die korrekte Implementierung spielt natürlich auch für den Ingenieur, also den Informatik-Anwender in der Industrie, eine sehr bedeutende Rolle. Sie bildet aber nicht sein einziges, oft nicht einmal sein schwerstes Problem. Dafür gibt es zwei Gründe:

- Die Erfahrungen zeigen, dass sich in sehr vielen Anwendungen (ausser in solchen mit hohem Risiko) auch mit fehlerhafter Software leben lässt. Die Aussage, ohne Korrektheit seien alle anderen Eigenschaften der Software irrelevant (nicht ganz so scharf formuliert bei Goos, 1976, S. 48), stimmt eben nicht: Nur die wenigsten Anwender würden von ihrem definitiv fehlerhaften Betriebssystem auf ein sicher korrektes umsteigen, wenn dieses um Grössenordnungen langsamer wäre. Auch die Korrektheit hat also ihren endlichen Wert, und der ist nicht so hoch, wie die Theorie das gelegentlich postuliert.

- Wichtiger noch ist die besondere Schwierigkeit, unter den Bedingungen der Praxis die Anforderungen an ein System so festzustellen, dass daraus eine brauchbare Spezifikation entsteht. "Bedingungen der Praxis" bedeutet hier:
  - Die Spezifikation muss nach begrenzter Zeit vorliegen.
  - Die Randbedingungen sind nur teilweise klar.
  - Der Kunde, der die Spezifikation akzeptieren muss, hat keine Ausbildung als Informatiker.
  - Selbst, wenn zum Spezifizieren ein hochqualifizierter Mitarbeiter zur Verfügung steht (ein seltener Ausnahmefall) und damit der Einsatz einer streng formalen Sprache möglich ist, muss das Ergebnis doch verständlich sein für die Codierer.
  - Das System wird über einen langen Zeitraum (typisch 5 bis 20 Jahre) modifiziert und erweitert werden, wobei nicht a priori klar ist, worin diese Aenderungen bestehen werden (vgl. Lehman, 1980).

Aus diesem Gründen erscheinen die Schwierigkeiten der korrekten Implementierung einer Spezifikation zwar nicht als gering, aber als geringer. Dieser Beitrag ist aus der Sicht des Ingenieurs geschrieben, und er konzentriert sich daher auf die Frage, wie sich die Anfertigung einer Spezifikation erleichtern lässt.

Lösungsneutrale und konstruktive Spezifikation

Rein formal stellt die Spezifikation ein Prädikat dar, das auf alle Systeme angewandt werden kann und genau dann den Wert WAHR hat, wenn das System der Spezifikation genügt, also eine zulässige Implementierung darstellt. Spezifikationen, die sich

ganz konkret so interpretieren lassen, nennt man überprüfbar. (Das Wort ist eigentlich falsch, denn überprüft wird das Programm anhand der Spezifikation.) Typisch sind Forderungen wie

- Das Programm soll den GGT zweier natürlicher Zahlen berechnen.
- In keinem Fall darf eine Tastatur-Eingabe einen Laufzeitfehler verursachen.
- Die Antwortzeit darf 1 s niemals übersteigen, solange die Hardware des Rechners fehlerfrei arbeitet.

Nehmen wir einmal an, dass die Spezifikation nur Aussagen dieser Art enthält, so handelt es sich um eine Black-Box-Spezifikation. Das Zielsystem wird als ein amorpher Automat beschrieben, von dem nur das Ein-/Ausgabe-Verhalten betrachtet werden kann. Noch vor wenigen Jahren war es in der Literatur unbestritten, dass jede Spezifikation so aussehen müsse, und es galt nur als eine Frage der Zeit, bis wir die geeigneten Verfahren und Sprachen dafür hätten.

Diese Hoffnung war unbegründet, denn es hat sich herausgestellt, dass die strenge Einhaltung der Regel, die Spezifikation solle das "WHAT, not HOW" enthalten, in der Praxis weder möglich noch erwünscht ist (Swartout, Balzer, 1982; Ludewig, 1982). Daher gehen heute die meisten Ansätze davon aus, dass die Spezifikation wenigstens teilweise den Charakter eines Modells hat, über dessen Struktur und andere innere Eigenschaften Aussagen zulässig und sogar notwendig sind.

Die beiden Ansätze sollen hier als lösungsneutrale und als konstruktive Spezifikation bezeichnet werden. Der wesentliche Unterschied ist, dass die konstruktive Spezifikation Aussagen über Teile des Gesamtsystems enthält, die prinzipiell nicht erforderlich sind und daher in der lösungsneutralen Spezifikation vermieden werden.

Betrachten wir als Beispiel die folgende lösungsneutrale Spezifikation:

> Zu liefern ist ein Strassen-Transportmittel für genau eine Person, die dieses Transportmittel aus eigener Kraft, also ohne Fremdenergie, antreibt. Die Person kann dazu eine Leistung bis zu 80 mW liefern. Das Transportmittel soll höchstens 60 cm breit und 150 cm lang sein und nicht schwerer als 20 kg sein. Es soll eine Gesamtlebensdauer von 10 Jahren und 20 000 km haben und nicht mehr als 300 SFr. kosten.

Eine konstruktive Spezifikation zur gleichen Aufgabe könnte etwa lauten:

Zu Liefern ist ein Fahrzeug, bestehend aus einem Rahmen, zwei Rädern mit Bereifung, Antrieb über Pedale und Kette, Sattel, Lenker und Gepäckträger. Die Teile sind in bestimmter Weise angeordnet: ... .
Gewicht, Lebensdauer, Preis, Grösse sind wie folgt beschränkt: ... .

Vergleich der lösungsneutralen und der konstruktiven Spezifikation

Die lösungsneutrale Spezifikation hat, wie der Name sagt, den Vorteil völliger Neutralität gegenüber verschiedenen Lösungsvarianten. Dadurch sind auch Lösungen möglich, die sich fundamental von den zuvor bekannten unterscheiden.

Die konstruktive Spezifikation enthält bereits Entwurfselemente und ist daher (wenigstens prinzipiell) operational, also simulierbar, falls die Grundelemente operational sind. Entsprechend ist sie auch leichter zu implementieren.

Die Verwendung anschaulicher Grundelemente macht auch die Spezifikation anschaulich, und das selbst dann, wenn die Grundelemente nicht präzise definiert sind. Im Beispiel oben gilt dies für Wörter wie "Rahmen" und "Rad", in der bekannten Spezifikationssprache PSL (Teichroew, Hershey, 1977) für Begriffe wie INPUT, PROCESS und OUTPUT, die nirgends definiert sind. Daher kann eine konstruktive Spezifikation trotz Unklarheiten erstellt werden.

Der wichtigste Vorteil der konstruktiven Spezifikation zeigt sich jedoch beim Versuch, ein sehr komplexes System in den Griff zu bekommen: Nur durch die Darstellung mittels bekannter (oder doch vorstellbarer) Grundelemente erhalten wir eine Vorstellung des Systems und werden dadurch erst in die Lage versetzt, sinnvolle Forderungen (oder Fragen) zu stellen. Dies ist besonders wichtig im Hinblick auf die völlig normale Situation, dass der Kunde erst im Zuge der Entwicklung seine Anforderungen klärt. Die in der Literatur gezeigten lösungsneutralen Spezifikationen sind in aller Regel auf sehr kleine Systeme beschränkt (beispielsweise auf den unsäglichen Stack).

Wir sind daher der Meinung, dass es bei Arbeiten an komplexen Systemen zur konstruktiven Spezifikation keine Alternative gibt.

Eine konstruktive Spezifikation soll nachfolgend als Modell bezeichnet werden. Hier geht es also um einen anschaulichen Modellbegriff, wie wir ihn auch bei Modellen von Bauwerken anwenden. Natürlich hat auch eine integrale Spezifikation Modell-Charakter, doch geschieht dort die Modell-Bildung nur durch Abstraktion.

Ebenen der Modellierung

Der Modell-Begriff, wie er hier verwendet wird, ist mehrstufig, wie von Luft (1984) beschrieben. Ist die spezielle Spezifikation also ein Modell des zu entwickelnden Systems, so sind die Spezifikationen einer gewissen Klasse wiederum repräsentiert durch ein (Meta-)Modell, das typisch als Spezifikationssprache ausgeprägt ist. Die - in den meisten Veröffentlichungen übliche - Konzentration auf eine Sprache führt allerdings weg vom Wesentlichen: Während im Sinne der Modellbildung die Syntax irrelevant ist (bis hin zur Frage, ob es sich um eine lineare oder eine zweidimensionale Syntax handelt), sind in der Sprache die für die Modellbildung zentralen Konzepte - dieses Wort soll nachfolgend die Meta-Modelle bezeichnen - nicht unmittelbar erkennbar.

Ueber den Konzepten gibt es noch eine weitere Ebene, auf der die Strukturen der Konzepte definiert sind. Anschaulich ist dies die Ebene der Datenbank, auf der ein Spezifikationssystem realisiert ist. Die Grundbegriffe sind dann durch das Datenbank-Modell repräsentiert, beispielsweise das Entity-Relationship-Prinzip. Die Konzepte (im Sinne der Definition oben) werden durch ein spezielles Datenbank-Schema beschrieben, und die Spezifikation schliesslich bildet ein spezielles Modell im Rahmen des Spezifikationssystems (siehe Abb. 1).

Ebene	repräsentiert durch
Konzept-Struktur	Datenbank-System
(Darstellungs-) Konzept	DB-Schema, Semantik der Spezifikationssprache
Modell (des Zielsystems)	Spezifikation
Gegenstand	spezifizierte Software

Abb. 1: Modellierungsebenen

Nicht-exakte Modellierung

Auch beim Aufbau eines Software-Systems aus einer Baustein-Bibliothek erzeugen wir ein Modell im oben definierten Sinne, vorausgesetzt, die verwendeten Bausteine sind anschaulich. Ein Beispiel sind Programmiersysteme auf der Basis von Funktions-

blocksprachen (Kaufmann, Schillinger, 1984). Die Bibliothek stellt dabei eine vollständig implementierte Menge von Konzepten dar. Ihre Bedeutung ist zumindest durch die Implementierung präzise und eindeutig definiert, es entsteht also ein formales Modell, das im genannten Beispiel auch operational ist. Gewisse Spezifikationsmethoden liefern ebenfalls ein formales Modell, beispielsweise die denotationelle Spezifikation (Klaeren, 1983, S. 17), das aber in der Regel nicht operational ist.

Die Vorteile der formalen Spezifikation liegen auf der Hand: Sie ist perfekt im Sinne des oben beschriebenen Mathematiker-Standpunkts. Ihr wesentlicher - und für den Ingenieur entscheidender - Nachteil liegt in der schwierigen Handhabung. Fragt man einen Techniker, wie dies oder jenes funktioniert, so zeichnet er in der Regel eine Skizze auf, die einige wichtige Komponenten oder Merkmale zeigt. Die Skizze hat wieder Modell-Charakter, doch ist dieses Modell nicht exakt, es lässt sich nicht nach klaren Regeln auf das skizzierte System übertragen. Die Abstraktion in einer Skizze ist von drei Arten:

- Fehlen ganzer Teile (Beschränkung auf das Wesentliche, Einengung)
- Unbestimmtheiten (offene Entscheidungen, Vagheit)
- Fehlen von Details (Abstraktion im engeren Sinne, Vereinfachung)

Hinzu kommen noch Unklarheiten, die typisch aus einer Mischung der drei genannten Abstraktionsarten entstehen und sich in schwammigen Formulierungen ausdrücken.

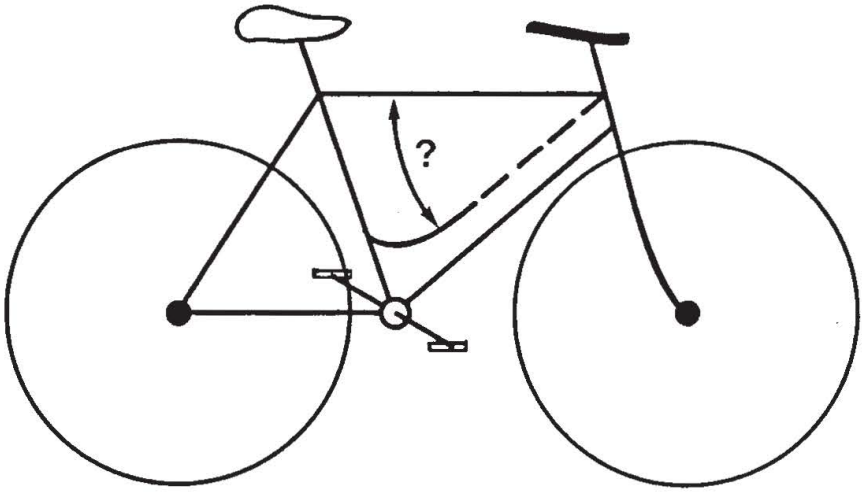

Abb. 2: Beschreibung durch eine Skizze

Für das Beispiel "Fahrrad" sieht eine Skizze typisch aus wie Abb. 2. Darin fehlen einige Komponenten (Bremsen, Beleuchtung), und Entscheidungen bleiben offen; letzteres zeigt sich in der Skizze durch vergröbernde Unschärfe (Lenkerform ist

nicht erkennbar) oder durch mehrere Varianten (für alternative Rahmenformen). Schliesslich fehlen Details (Felgen, Speichen und Reifen als Teile der Räder).

Ein halbformales Modell

Eine konstruktive Spezifikation in der Art einer Skizze bezeichnen wir als halbformales Modell. Die zugrundeliegenden Konzepte müssen dazu Freiheit für die oben genannten Abstraktionsformen bieten. Für Fahrräder wären dies typisch Begriffe wie "Rahmen", "Rad" usw. Für Echtzeitprogramme bietet unser Spezifikationssystem SPADES (Ludewig et al., 1985) in Anlehnung an die Denkwelt von PSL/PSA (Teichroew, Hershey, 1977) einige relativ einfache Konzepte an, nämlich

Module	zur Beschreibung der statischen Struktur
Aktoren (Prozeduren, Blöcke)	als ausführbare Komponenten
Parameter	in Verbindung mit Prozeduren
Medien (Variablen, Puffer, Trigger, Betriebsmittel)	für die Kommunikation zwischen Aktoren
Typen	zur näheren Beschreibung von Medien
Fristen	zur Beschreibung dynamischer Eigenschaften
Konstanten	zur Angabe von System-Parametern
Textobjekte	zur Archivierung (noch) nicht formalisierter Angaben und für Dokumentationszwecke

Objekte dieser Arten lassen sich miteinander verknüpfen, so dass die folgenden Beziehungen beschrieben werden können:

strukturelle Eigenschaften,	z.B. Modulhierarchie, Typaufbau
Kommunikation,	z.B. lesen und schreiben von Variablen
Koordination,	z.B. starten oder verzögern von Aktoren
Abläufe,	z.B. sequentielle Ausführung von Aktoren
Restriktionen,	z.B. für den Gültigkeitsbereich eines Mediums
allgemeine Verweise	zwischen beliebigen Objekten

Aus Instanzen dieser Konzepte können Spezifikationen aufgebaut werden. Es entsteht damit zunächst eine "abstrakte Skizze". Es liegt nahe, diese auch konkret als Bild zu behandeln, wie es die Abb. 3 zeigt.

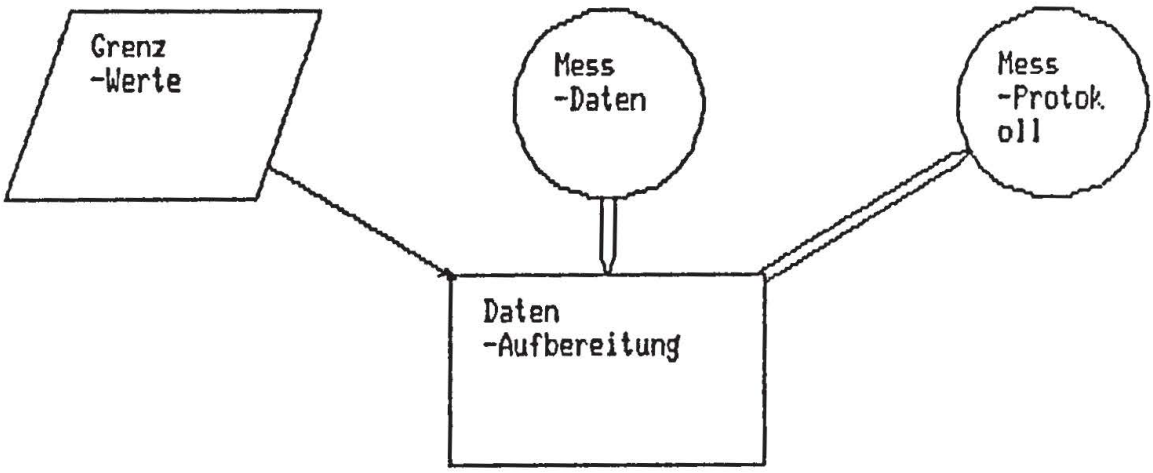

Abb. 3: Beispiel einer Software-Skizze (Datenfluss)

Bei der Auswahl von Konzepten für ein Spezifikationsproblem gibt es ein Dilemma: Einerseits soll die Zahl klein gehalten werden, um die Einarbeitung und Anwendung zu erleichtern und damit auch die Bereitschaft zur Anwendung des Systems zu erhöhen. Andererseits reichen in fast keinem speziellen Fall die wenigen Konzepte zu einer exakten Spezifikation aus, so dass es naheliegt, ihre Zahl immer weiter wachsen zu lassen. Zwei Gegenmittel stehen uns dafür zur Verfügung:

- Die Sprache wird auf einen bestimmten Anwendungsbereich (SPADES: Echtzeit-Software) zugeschnitten. Damit fallen Anforderungen aus anderen Bereichen weg.

- Da die Modellierung nicht exakt sein muss, hat der Anwender die Möglichkeit, sich die Begriffe "hinzubiegen", indem er Erklärungen in Textform zufügt, die seine spezielle Interpretation angeben. Auf schematischen technischen Zeichnungen beispielsweise werden Symbole (für "Welle", "Lager", "Zahnrad" usw.) durch Schrift ("Wälzlager", "schrägverzahnt") ergänzt. Auch SPADES bietet formal nur ein kleines Vokabular an, das jedoch informal präzisiert wird. Man kann dies vergleichen mit einem Baukasten, der nur wenige Grundformen bietet. Will man daraus das Modell eines Hauses bauen, so kann man die Elemente "kommentieren", etwa in der Form "Die Neigung des Daches ist real etwas geringer als im Modell".

Anwendung des halbformalen Modells

Wir müssen das halbformale Modell auf der Grundlage des informalen Modells aufbauen, das wir zu Beginn im Kopf haben. Dazu benötigen wir gewöhnlichen Text oder einfache, durch Text kommentierte Graphen. Anschliessend können wir beginnen, das Modell genauer zu machen, indem wir Fehlendes ergänzen, offene Fragen klären und Details zufügen.

Entsprechend beginnt in SPADES die Spezifikation in aller Regel mit sogenannten Text-Objekten. Diese können hierarchisch gegliedert werden, so dass bereits eine gewisse Strukturierung erkennbar wird. Im Laufe der Zeit werden immer mehr Informationen aus den Texten "herausgebrochen" und in spezifische Objekte, z.B. Module oder Prozeduren, umgewandelt. Dieser Prozess wird durch das Werkzeug unterstützt. Freilich bleiben die Texte dabei ein unverzichtbarer Bestandteil der Spezifikation, schon allein, weil gewisse Aussagen nicht formalisiert werden können (beispielsweise alle Arithmetik). Mit Hilfe der Werkzeuge wird die Spezifikation gespeichert, überprüft und bei Bedarf in andere Darstellungsformen gewandelt. Ist der Anwender der Meinung, alle zur Implementierung notwendigen Informationen in der Spezifikation (d.h. in der Datenbank) gesammelt zu haben, so lässt er nochmals deren formale Konsistenz überprüfen und übergibt dem Codierer die vom Spezifikationssystem generierte Dokumentation.

Konzept-Struktur und Datenbank-System

Was oben für die Vielfalt der Konzepte gesagt wurde, gilt ganz gleich auch auf der nächsthöheren Ebene, auf der Ebene der Konzept-Strukturen. Ist die Konzept-Struktur zu primitiv, so lassen sich gewisse Konzepte nicht beschreiben (z.B. erlaubt die streng hierarchische Struktur keine Beschreibung der Aufrufbeziehungen mehrfach verwendeter Unterprogramme, da diese keinen Baum bilden). Eine grosse Struktur-Vielfalt (z.B. Bäume und Schichten-Strukturen und allgemeine gerichtete Graphen, deren Knoten wieder strukturiert sein dürfen) erschwert dagegen das Verständnis und mehr noch die Realisierung der Datenbank, in der die Beschreibungen gespeichert werden sollen.

Die allgemein zu beobachtende Neigung, komplexe Systeme durch Objekte und ihre Verbindungen darzustellen, legt es nahe, diese Begriffe auch zur Grundlage der Konzept-Struktur zu machen. In diesem Sinne schuf Chen (1976) das Entity-Relationship-Konzept (das nachfolgend, um eine Verwechslung mit dem oben eingeführten Konzept-Begriff zu vermeiden, als ER-Prinzip bezeichnet wird). Das ER-Prinzip ist einerseits für die Konzepte im wesentlichen ausreichend, andererseits

auch als Datenbank-Modell für eine effiziente Implementierung geeignet. Es wurde damit zur bevorzugten Konzept-Struktur für Spezifikationssysteme. Auch die Datenhaltung in ESPRESO, dem Vorgänger von SPADES, wurde nach dieser Idee realisiert (und sehr pragmatisch ausprogrammiert). An einigen Punkten gab es jedoch Schwierigkeiten, beispielsweise bei der Behandlung der Texte, für die im ER-Prinzip kein geeigneter Platz erkennbar war.

SPADES verwendet daher eine (ebenfalls selbst realisierte) Datenbank (Glinz, Huser, Ludewig, 1985), die in einigen Merkmalen über das ER-Prinzip hinausgeht. So besteht die Möglichkeit zur Generalisierung von Arten und Relationen (und damit zur Formulierung der Unbestimmtheit) und zur Bildung von Mustern, die sowohl eine elegante Beschreibung von Aehnlichkeiten zwischen verschiedenen Teilen der Spezifikation wie auch eine Darstellung von Varianten gestatten. Eine Erweiterung des Konzepts für die Versionenverwaltung ist in Arbeit. Die Datenbank bietet damit wesentlich mehr Flexibilität, als durch SPADES gegenwärtig ausgenutzt wird.

Erfahrungen und Probleme

Eine Reihe von Erfahrungen mit der halbformalen Spezifikation gehört inzwischen zur Folklore des Software Engineerings, vgl. etwa Reinshagen (1983, 141 ff.). Hier sollen unsere eigenen auf den Modellierungsaspekt bezogenen Beobachtungen beim praktischen Einsatz von SPADES wiedergegeben werden.

- Die Anwendung der Konzepte, im Klartext also der Einsatz einer grundsätzlich neuen Sprache, erfordert massive Schulung, denn der Sprung ist viel grösser als beim Wechsel auf eine neue Programmiersprache.

- Allgemein wird es als vorteilhaft empfunden, eine wenigstens teilweise formalisierte und damit durch Werkzeuge analysierbare Darstellung zu besitzen, lange bevor an eine vollständige Formalisierung oder Realisierung zu denken wäre. Die halbformale Spezifikation konkurriert also erfolgreich mit der Spezifikation in natürlicher Sprache (und nicht mit der Codierung).

- Ein echtes Problem ist die Tendenz, die Spezifikationssprache als Codiersprache zu missverstehen oder, wenn die Anwender aus dem Hardware-Bereich kommen, die Strukturierung als Geräte-Entwurf aufzufassen. Hier wird quasi der neue Schraubenzieher nicht verwendet, um damit wie geplant Schrauben zu drehen, sondern um (wie bisher mit dem Hammer) Nägel einzuschlagen.

- Eine Schwierigkeit ist die Abgrenzung der Anwendungsphase. Die Frage, wann man mit dem "Füttern" des Spezifikationssystems beginnen, wann man aufhören sollte, wird sich nach unserer Meinung niemals ganz klar beantworten lassen; wir empfehlen den Einsatz des Spezifikationssystems von Anfang des Projekts bis zu einer Situation, in der die Konzepte nicht mehr ausreichen.

- Auch die Anwendungsbreite ist unklar, also die Frage, welche Bereiche der Spezifikation durch die Konzepte sinnvoll abgedeckt werden können. Bei uns sind solche Grenzbereiche die Dialoge und die Kommunikationsprotokolle. Prinzipiell gibt es auf den Wunsch nach einem grösseren Sprachumfang drei Antworten, jede mit offensichtlichen Vor- und Nachteilen:

  • strikte Beschränkung auf eine "kleine" Sprache

  • Ausbau der Sprache durch Konzepte für spezielle Aspekte

  • Unterstützung auch anderer Sprachen (durch Definition von Schnittstellen und durch Werkzeuge), so dass in einer Spezifikation mehrere Sprachen eingesetzt werden können. Für unsere Anwendung von SPADES wäre beispielsweise eine Kombination mit SDL (Specification and Description Language des CCITT) zu erwägen.

- Das Entity-Relationship-Prinzip beruht auf der Darstellung durch einen bipartiten Graphen. Eine auch wirklich graphische Darstellung ist also naheliegend und wird von den Anwendern im technischen Bereich dringend gefordert. Die entsprechende Erweiterung von SPADES wurde vor kurzem fertiggstellt; Abb. 3 ist damit generiert worden (vgl. Ludewig et al., 1985).

Ausblick

Diese Arbeit ist zu verstehen als Versuch, auf dem - in weiten Teilen noch immer brachliegenden - Feld der Entwicklungsmethoden voranzukommen. Wir hoffen, damit die Aufmerksamkeit von der Sprach-Verarbeitung, also von den Werkzeugen, auf die Sprach-Inhalte zu lenken und so eine Diskussion darüber anzustossen, welche Konzepte für die Spezifikation besonders geeignet sind, in welchem Masse sie nach Anwendungen aufgefächert werden müssen und wie weit sie dem menschlichen Bedürfnis nach unscharfer Darstellung entgegenkommen können.

Literaturangaben

Chen, P.P.-S. (1976): The Entity-Relationship Model - toward a unified view of data. ACM Transactions on Data Base Systems, 1, 1, 9-36.

Glinz, M., H.J. Huser, J. Ludewig (1985): SEED - A database system for software engineering environments. in Blaser, Pistor (Hrsg.): Datenbanksysteme für Büro, Technik und Wissenschaft, Informatik-FB 94, Springer, S.121-126.

Goos, G. (1976): Programmkonstruktion. Skriptum, Universität Karlsruhe.

Kaufmann, F., D. Schillinger (1984): Funktionale Sprache als anwenderfreundliches Projektierungsmittel. Brown Boveri Mitteilungen, 71, 488-493.

Klaeren, H.A. (1983): Algebraische Spezifikation. Springer Verlag, Berlin usw.

Lehman, M.M. (1980): Programs, life cycles, and laws of software evolution. Proc. of the IEEE, 68, 9, 1060-1076.

Ludewig, J. (1982): Computer aided specification of process control software. IEEE COMPUTER, Mai 1982, 12-20.

Ludewig, J., M. Glinz, H.J. Huser, G. Matheis, H. Matheis, M.F. Schmidt (1985): SPADES - A specification and design system and its graphical interface. 8th Intern. Conf. on Software Engineering, IEEE, London, August 1985.

Luft, A.L. (1984): Zur Bedeutung von Modellen und Modellierungs-Schritten in der Software-Technik. Angewandte Informatik, 5, 189-196.

Reinshagen, K.-P. (1983): Erfahrungen beim Einsatz eines entwurfsunterstützenden Spezifikationssystems. In J. Ludewig: Spezifikation von Realzeit-Systemen - Konzepte, Lösungen, Erfahrungen. Schweizerische Gesellschaft für Automatik, Seefeldstr. 301, 8008 Zürich. pp.127-152.

Swartout, W., R. Balzer (1982): On the inevitable intertwining of specification and implementation. Commun. ACM, 25, 7, 438-440.

Teichroew, D., E.A. Hershey III (1977): PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems. IEEE Trans. Software Eng., SE-3, 41-48.