J. Ludewig (Hrsg.): 54. Tagung der SGA, Baden/Schweiz, März 1983, 1-27

Stand der Forschung und Technik auf dem Gebiet der rechnerunterstützten Spezifikation von Prozessrechner-Software

Jochen Ludewig, Brown Boveri Forschungszentrum, Baden/Schweiz

Zusammenfassung

Das Thema "Spezifikation" wurde von den Informatikern identifiziert und bearbeitet im Zusammenhang mit der Entdeckung des Gebietes "Software-Engineering". Dies gilt auch für die Spezifikation solcher Systeme, die nur teilweise aus Software bestehen oder bei denen die Unterscheidung zwischen Hard- und Software noch nicht gemacht wurde. Daher soll hier versucht werden, das Gebiet von der Software-Spezifikation her anzugehen.

Nach einem kurzen Abriss der Motivation für Spezifikationssysteme und ihrer Geschichte werden die in diesem Zusammenhang wichtigen Begriffe diskutiert. Auch die übliche Einteilung des Software-Entstehungsprozesses, der "software-life-cycle", wird angesprochen.

Für die einzelnen Komponenten der Spezifikationssysteme haben sich Lösungen herauskristallisiert, die in vielen Systemen zu finden sind. Diese werden gegenübergestellt und diskutiert. Dann werden der aktuelle Stand und die Trends skizziert. Am Schluss stehen einige Erfahrungssätze zur Einführung eines Spezifikationssystems.

Gliederung

1. Motivation und Geschichte der Spezifikationssysteme

Im Verlauf der sechziger Jahre zeichnete sich - zunächst in den USA - die sogenannte Software-Crisis ab. Grosse Projekte (im militärischen Bereich) waren gescheitert, weil sie begonnen worden waren nach dem Vorbild kleiner, überschaubarer Projekte. Allgemein wurden Fertigstellungstermine und Kostenpläne weit überschritten.

In diesem Zusammenhang fanden die Aussagen von Boehm (1976) über die Bedeutung der Spezifikation grosse Beachtung. Insbesondere die (noch immer gültige) Feststellung, dass ein Fehler um so teurer ist, je länger er im Projekt vorhanden ist und die weitere Entwicklung beeinflusst (Abbildung 1), brachte der systematischen Spezifikation viele Anhänger. Ein weiterer Grund war und ist die Schwierigkeit aller Software-Hersteller, sich vor der Implementierung mit dem Kunden über das Produkt zu verständigen.

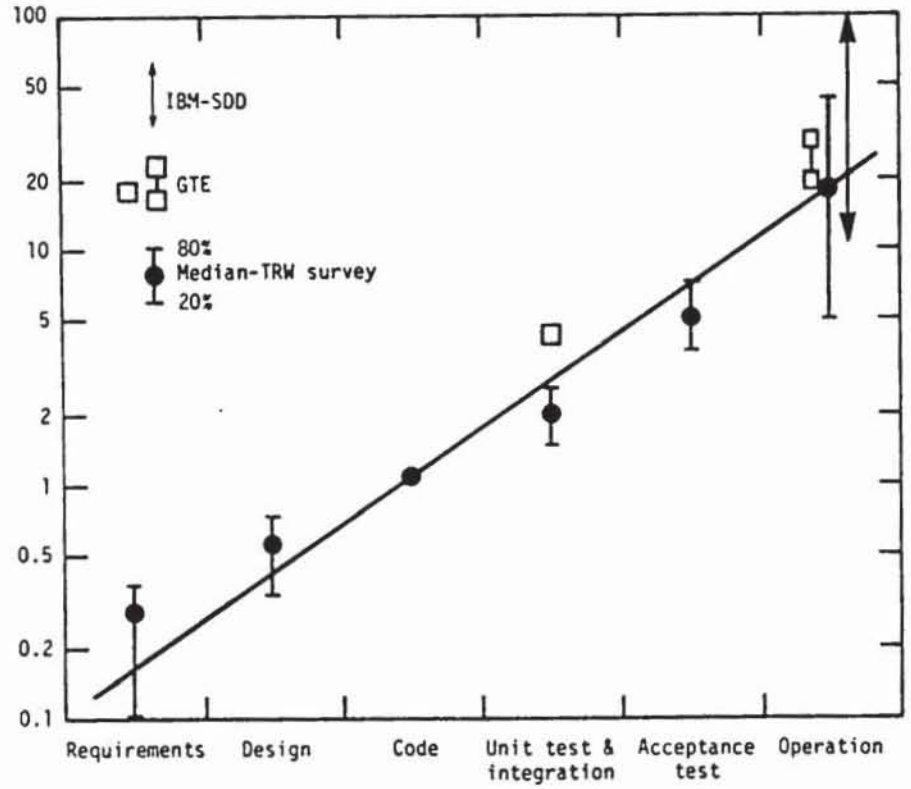


Abb. 1: Mittlere Kosten von Fehlern über der Entdeckunsphase, nach Statistiken von IBM, GTE und TRW (aus Boehm, 1976, Fig. 3)

Der geschilderte Bedarf an Unterstützung während der Spezifikationsphase traf Anfang der siebziger Jahre mit ersten Versuchen zusammen, die Beschreibung von Systemen vor Eintritt in die Codierung mit Rechnern zu unterstützen. Das ISDOS-Projekt an der University of Michigan, Ann Arbor, hat in dieser Entwicklung mit dem PSL/PSA-System eine zentrale Rolle gespielt (Teichroew, Sayani, 1971; Teichroew, Hershey, 1977).

Unter den vielen Einflüssen auf die Entwicklung im Gebiet der Spezifikation nehmen die neueren, sehr "hohen" Programmiersprachen einen wichtigen Platz ein.

2. Terminologie

Denn grad wo die Begriffe fehlen,
da stellt das Wort zur rechten Zeit sich ein.
Mephisto in Goethes Faust I

Zunächst sind einige Begriffsklärungen notwendig, denn im Software-Engineering ist die Terminologie sehr unklar. Dies beginnt mit dem Begriff "Software-Engineering" selbst; er wurde nicht als Beschreibung, sondern als Provokation geboren (als Bezeichnung für etwas, das es nicht gibt, aber geben sollte, F.L. Bauer, 1972). Durch die fortwährende Entwicklung hat sich bisher kein einheitlicher Sprachgebrauch durchsetzen können, und wir werden mit diesem Zustand noch einige Jahre leben müssen. Verstärkt wird diese Verwirrung, wenn ein Begriff in Mode kommt und für beliebige Inhalte verwendet wird. Dies ist gerade mit dem Wort "Spezifikation" in den vergangenen Jahren geschehen.

Hier soll der Spezifikationsbegriff so verwendet werden, wie es in anderen Ingenieur-Disziplinen üblich ist: Eine Spezifikation ist die Beschreibung eines Gegenstandes unter einem bestimmten Aspekt; insbesondere ist dies der Aspekt des Kunden.
In Uebereinstimmung mit der üblichen Interpretation wird hier in erster Linie von derjenigen Spezifikation gesprochen, die der Verständigung zwischen dem Anbieter und dem Kunden dient. Natürlich sind darin firmeninterne Kunden und Pseudo-Kunden, etwa eine Marketing-Abteilung, die die Kundenwünsche formuliert, eingeschlossen.

2.1 Lebenslauf-Modelle

Versuche, die bei der Software-Entwicklung anfallenden Arbeiten zu differenzieren und Phasen im zeitlichen Ablauf zu unterscheiden, haben zu sogenannten Lebenslauf-Modellen ("life-cycle models") geführt. Solche Modelle wurden zunächst als einfache Kette von Arbeitsschritten dargestellt (Teichroew, Sayani, 1971; Hice, Turner, Cashwell, 1974). Abgesehen von kleineren Unterschieden haben alle Gliederungen etwa die folgende Form:

Aufgabenstellung
System-Analyse
Anforderungsspezifikation
Systementwurf
Codierung und Test
Integration
Installation
Betrieb mit Korrekturen und Anpassungen

Solche Darstellungen lassen sich unter drei Aspekten interpretieren, nämlich als Phasen-Modell (Terminplanung und Personaleinsatz), als Beschreibung der Tätigkeiten und als Beschreibung der (Teil-)Ergebnisse. Bei naiver Betrachtung fallen alle drei zusammen. In der Praxis entsteht ein Konflikt, der zu Brüchen führt:
Die Spezifikation (als Tätigkeit) kann nicht beendet, die Spezifikation (als Dokument) nicht abgeschlossen werden, bevor gewisse Entwurfsentscheide gefällt sind. Geht man also vom Phasen-Modell aus, wie es zur Projekt-Kontrolle erforderlich ist, so liegen die Tätigkeiten "spezifizieren" und "entwerfen" nicht mehr sauber getrennt in den jeweiligen Phasen. Umgekehrt führt die Orientierung an Tätigkeiten dazu, dass das Modell nicht mehr zur Projektüberwachung taugt. Aenliches gilt für die anderen Phasen-Grenzen.

Die beschriebenen Probleme haben zu Modifikationen des Modells geführt (siehe z.B. Boehm, 1976, Fig. 2), die jedoch in keinem Fall den grundlegenden Konflikt beseitigt haben (vgl. Ludewig, 1982a; McCracken, Jackson, 1982). Daher scheint es vorteilhaft, die Interpretation auf einen einzigen Aspekt zu begrenzen. Dazu bietet sich die ursprüngliche Anwendung an, das Phasenmodell für das Software-Management. (Speziell zur Abgrenzung zwischen Spezifikation und Entwurf siehe 2.3)

2.2 Das System-Dreieck

In vielen Diskussionen über Spezifikationssysteme entstehen Missverständnisse, weil Konzepte, Methoden, Sprachen und Werkzeuge durcheinandergeworfen werden. Das System-Dreieck (Abbildung 2) soll den Zusammenhang veranschaulichen und damit die Unterscheidung erleichtern.

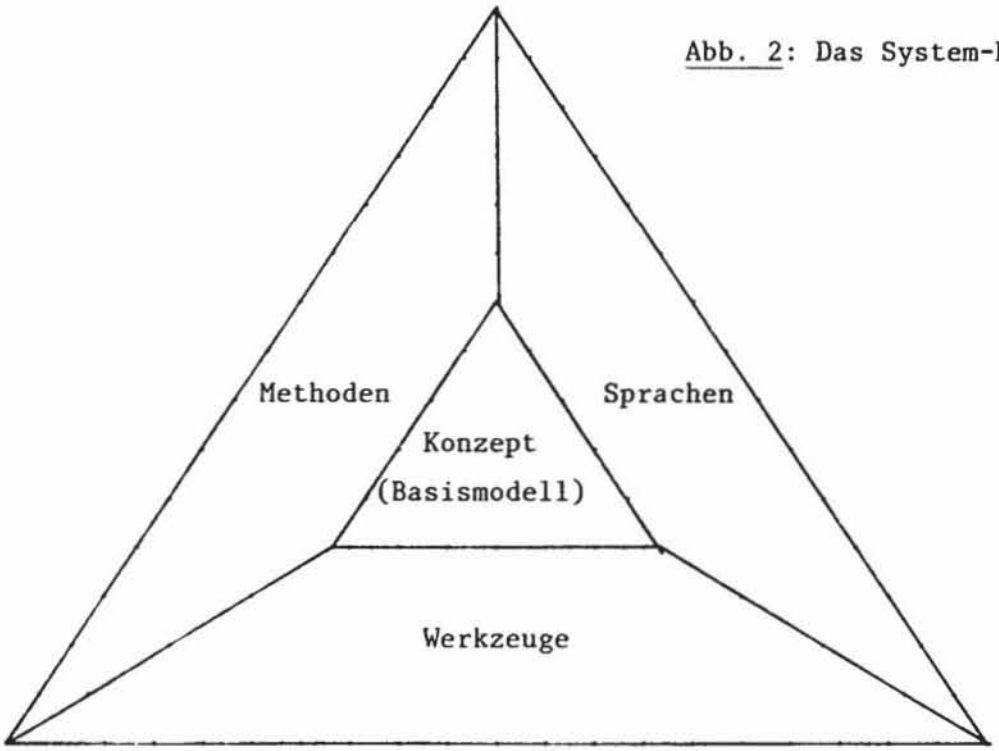


Abb. 2: Das System-Dreieck

Jedem Spezifikationssystem (wie auch jedem anderen Beschreibungssystem, z.B. einem Programmiersystem) liegen gewisse Konzepte zugrunde, beispielsweise das der endlichen Automaten oder das des hierarchischen Aufbaus. Daher bilden die Konzepte den Kern des System-Dreiecks.

Die Konzepte werden ausdrückbar als Elemente der Spezifikationssprache, so wie das Konzept der alternativen Ausführung durch die IF-THEN-ELSE-Konstruktion in PASCAL ausgedrückt wird. Für die in den Konzepten bereits vorhandene Semantik wird also eine Syntax gewählt.
Auch graphische Beschreibungen (SADT) haben den Charakter einer Sprache. Sagt man von verschiedenen Sprachen, dass sie äquivalent sind, so bedeutet dies, dass sie den selben semantischen Kern haben, also die gleichen Konzepte darzustellen erlauben.

Da wir das Modell unseres Systems in Form einer Spezifikation nur schrittweise aufbauen können, stellt sich die Frage, welche Schritte in welcher Reihenfolge notwendig sind. Beispielsweise kann man die Beschreibung einer hierarchischen Struktur in der höchsten oder in der tiefsten Ebene beginnen, also top-down oder bottom-up vorgehen. Dies ist die Methode des Spezifikationssystems.

Werkzeuge unterstützen und kontrollieren den Benutzer, indem sie ihn von schematischen Arbeiten (kopieren, suchen, umordnen, übersetzen) entlasten, wenn er nach der vorgesehenen Methode verfährt; gleichzeitig stellen sie sicher, dass die Spezifikationssprache korrekt verwandt wird. Sie ermöglichen automatische Prüfungen, indem sie aus der Spezifikation wieder die Bedeutung herausziehen und diese auf angestrebte Eigenschaften, z.B. Konsistenz, prüfen.

Ein ideales Spezifikationssystem besteht aus den vier genannten Teilen; es basiert auf einem sorgfältig gewählten Modell; Sprache(n), Methode und Werkzeuge sind darauf und untereinander wohlabgestimmt. Reale Spezifikationssysteme genügen diesem Anspruch vielfach nicht, vor allem, weil unsere Kenntnisse über die Konzepte noch ungenügend sind und daher oft mehr probiert als geplant wird. Einzelne Komponenten sind häufig nur schwach ausgeprägt (meist die Methode) oder fehlen ganz (manchmal das Werkzeug).

2.3 Eigenschaften einer guten Spezifikation

Die Spezifikation soll nach Möglichkeit so verfasst sein, dass sie alle relevanten Informationen, aber nur diese, enthält, und zwar in einer Form, die für den jeweiligen Leser verständlich und überprüfbar ist (vergleiche - auch für andere Definitionen - Kramer, 1982). Sie soll also vollständig, knapp, verständlich und testbar sein. Ihr Inhalt soll konsistent sein, also keine Widersprüche aufweisen.

Problematisch ist dabei, dass die Relevanz natürlich subjektiv beurteilt wird. Der Kunde und oft auch der Anbieter haben in vielen Situationen die Tendenz, Dinge für wesentlich zu halten, die bis zur Realisierung offen bleiben könnten. Dadurch werden ohne ausreichende Grundlagen Festlegungen getroffen, die später zu Fesseln werden. Die griffige Formel "WHAT before HOW" drückt das Bestreben aus, diesen Fehler zu vermeiden. Praktisch gesehen ist aber die strikte Trennung von Spezifikation und Entwurf eine Fiktion (Swartout, Balzer, 1982).

Prozessrechner-Software weist gegenüber anderer Software, etwa im kommerziellen Bereich, einige Besonderheiten auf, die sich auch in der Spezifikation niederschlagen. Parallelität ist durch die Umgebung, den technischen Prozess, vorgegeben. Antwortzeiten sind kritisch. Das Prozessrechensystem hat eine ungewöhnlich breite Schnittstelle zum Prozess, in vielen Fällen über Tausende von Sensoren etc. Da das gesamte System, Prozess und Leittechnik, logisch eine Einheit bildet, ist es erforderlich, auch den Prozess zu beschreiben. Nur so werden die Abhängigkeiten erkennbar.

Die oben genannten Merkmale einer guten Spezifikation müssen überprüfbar sein; nur dann ist die Qualität objektivierbar. Eine formlose Spezifikation in Umgangssprache hat keine präzise Bedeutung und ist möglicherweise für viele Menschen unverständlich (oder schlimmer: missverständlich). Ihre Vollständigkeit und Konsistenz lassen sich nur intuitiv, nicht objektiv beurteilen.

Daher spielt die Formalisierung eine zentrale Rolle. Verschiedene Stufen lassen sich unterscheiden:

informal = in freier Form und Umgangssprache.

formatiert = umgangssprachlich, aber in fester Form (z.B. auf einem Formular).

formal = in einer wohldefinierten formalen Sprache ausgedrückt. "wohldefiniert" bedeutet hier, dass nicht nur die Struktur (Syntax) festgelegt ist, sondern auch die Bedeutung (Semantik). Programmiersprachen sind in diesem Sinne (weitgehend) formal, daher kann ein Uebersetzer das Programm im Wesentlichen so "verstehen" wie ein menschlicher Leser. Ausnahmen bilden dabei z.B. die frei gewählten Namen (z.B. für eine Programm-Variable "Steigwinkel"), denen der menschliche Leser eine Bedeutung beimessen kann, die Maschine jedoch nicht.

Die Erfahrungen haben gezeigt, dass einerseits informale Spezifikationen nicht ausreichen, andererseits jedoch formale äusserst schwer "aus dem Stand" erreichbar sind. Der Sprung vom Informalen zum Formalen überfordert bei komplexen Systemen den Menschen. Daher ist ein weiterer Begriff entstanden:

halbformal = vorwiegend formal, jedoch mit "Inseln" für formatierte Information darin. Praktisch sieht das so aus, dass die Beschreibung ähnlich wie in einer Programmiersprache erfolgt, jedoch auch informale Texte enthält. Diese gehören zur Spezifikation und werden (anders als Kommentare in Programmen) auch von den Werkzeugen verwaltet, aber nicht interpretiert, sind also - wie die oben genannten Namen - nur für den menschlichen Leser verständlich.
Eine etwas andere, weniger exakte Interpretation des Wortes "halbformal" bezeichnet damit Sprachen, deren Semantik selbst nicht definiert ist (z.B. PSL und SADT). Tatsächlich handelt es sich hier eher um eine - sehr straffe - Formatierung.

Auf allen vier Stufen gibt es Spezifikationsverfahren und -sprachen. Ausgangs- und Endstufe stehen a priori fest: Die ersten Ideen sind unvermeidlich informal und vage, das Ergebnis schliesslich ist ein formales Programm.

Unter den oben genannten Eigenschaften einer guten Spezifikation kommt die Korrektheit nicht vor, denn Korrektheit lässt sich nur in Bezug auf vorgegebene Kriterien beurteilen. Die Spezifikation stellt für die weiteren Dokumente (Entwurfsbeschreibung, Code) einen solchen Bezugspunkt dar und ist daher auch für eine Verifikation unentbehrlich, lässt sich aber - als erstes Glied der Kette - selbst nicht verifizieren.
Praktisch sprechen wir dennoch von einer falschen Spezifikation, weil wir gefühlsmässig unsere (nur gedachte) Vorstellung vom Zielsystem als Bezugspunkt anerkennen. Man muss sich allerdings klarmachen, dass damit die Grenze zwischen Informatik und Psychologie überschritten ist.

2.4 Ein einfaches Beispiel

Als sehr einfaches Beispiel soll hier die Spezifikation angegeben werden, die ein Kunde für einen einfachen Taschenrechner haben könnte.

Gewünscht ist ein Gerät zur Durchführung arithmetischer Operationen mit folgenden Eigenschaften:

Anzeige:	mindestens 8 Dezimal-Stellen, LCD-Technik;
Rechengenauigkeit:	mindestens 12 Dezimal-Stellen;
Operationsvorrat:	Grundoperationen, Quadratwurzel, Operandentausch;
Form:	etwa rechteckig, maximal 200 mm lang, 15 mm dick;
Preis:	maximal 40 Fr.

Der Stil dieser Spezifikation ist, wie in technischen Spezifikationen üblich, formatiert. Die Bedeutungen sind vielfach unklar; "etwa rechteckig" kann zu einem Streitpunkt werden, auch "Grundoperationen" ist dubios. (Gehört das Löschen dazu ?)
Die Spezifikation ist wohl präziser als übliche Kundenangaben. Dennoch ist sie in vieler Hinsicht unvollständig. So fehlen beispielsweise Gewicht, Mindestgrössen und Anordnung der Anzeige und der Bedienungselemente, Lebensdauer, Energieversorgung. In besonderen Fällen können auch Angaben nötig sein, die normalerweise exotisch erscheinen, z.B. die Empfindlichkeit gegen gewisse Strahlen.

Scheinbar selbstverständliche Anforderungen sind weggelassen: So ist etwa die Antwortzeit nicht erwähnt, obwohl sie für den Gebrauchswert wesentlich ist und je nach Anwendung zu Reklamationen führen kann.
Die Forderung nach LCD-Anzeige ist sehr wahrscheinlich unnötig. Wichtig ist ein niedriger Stromverbrauch (der nirgends spezifiziert ist) und eine gute Ablesbarkeit. Vielleicht gibt es eine dem Kunden unbekannte Art der Anzeige, die viel besser geeignet ist.

Bei einem Taschenrechner lassen sich die Unklarheiten im allgemeinen im Dialog mit einem Verkäufer ausräumen. Ist das Ergebnis dennoch unbefriedigend, so kann man schlimmstenfalls den Fehlkauf wegwerfen. Prozessrechensysteme sind zu komplex und zu teuer für diesen einfachen Ansatz. Es werden Verfahren und Hilfsmittel benötigt, eine brauchbare, die tatsächlichen Anforderungen des Kunden enthaltende Spezifikation zu erhalten.

3. Standardlösungen

3.1 Konzepte

Jede Beschreibung eines Gegenstandes stellt eine Abstraktion dar, es sei denn, die Beschreibung geschieht durch eine exakte Reproduktion. Durch Abstraktion entfallen Informationen, so dass die verbleibenden um so deutlicher werden. Die Abstraktion stellt also einen bestimmten Aspekt heraus. Eine Strassenkarte etwa hebt die Strassen hervor, indem sie nur sehr wenig andere geographische Information mitteilt. Das Basismodell der Strassenkarte besteht im wesentlichen aus Land- und Wasserflächen, Ortsbezeichnungen und Strassen. Geologische oder meteorologische Angaben sind weggelassen, sie würden nicht nur wenig nützen, sondern die Uebersicht wesentlich verschlechtern.

Das Basismodell eines Spezifikationssystems sollte ähnliche Eigenschaften aufweisen, also das Wichtige betonen und das Unwichtige unterdrücken. Allerdings ist hier das Problem recht kompliziert, so dass noch weniger als bei einer Landkarte entschieden werden kann, welches Modell unter gegebenen Bedingungen das Beste ist. Nachfolgend werden einige wichtige Modelle vorgestellt. In allen realen Systemen sind mehrere Modelle kombiniert.

3.1.1 Struktur, Hierarchie

Ein wichtiges Merkmal jedes komplexen Systems ist seine Struktur. Da unsere Fähigkeit, mit umfangreicher Information umzugehen, äusserst beschränkt ist, benötigen wir eine hierarchische Struktur, im einfachsten Fall die Zerlegung grosser Einheiten in kleinere Komponenten (Simon, 1962). Die hierarchischen Beziehungen in einem Programmsystem sind vielfältig (Daten, Prozeduren, Moduln usw., vgl. Goos, 1973; Parnas, 1974), und praktisch jedes Spezifikationssystem unterstützt die Darstellung dieser Beziehungen.

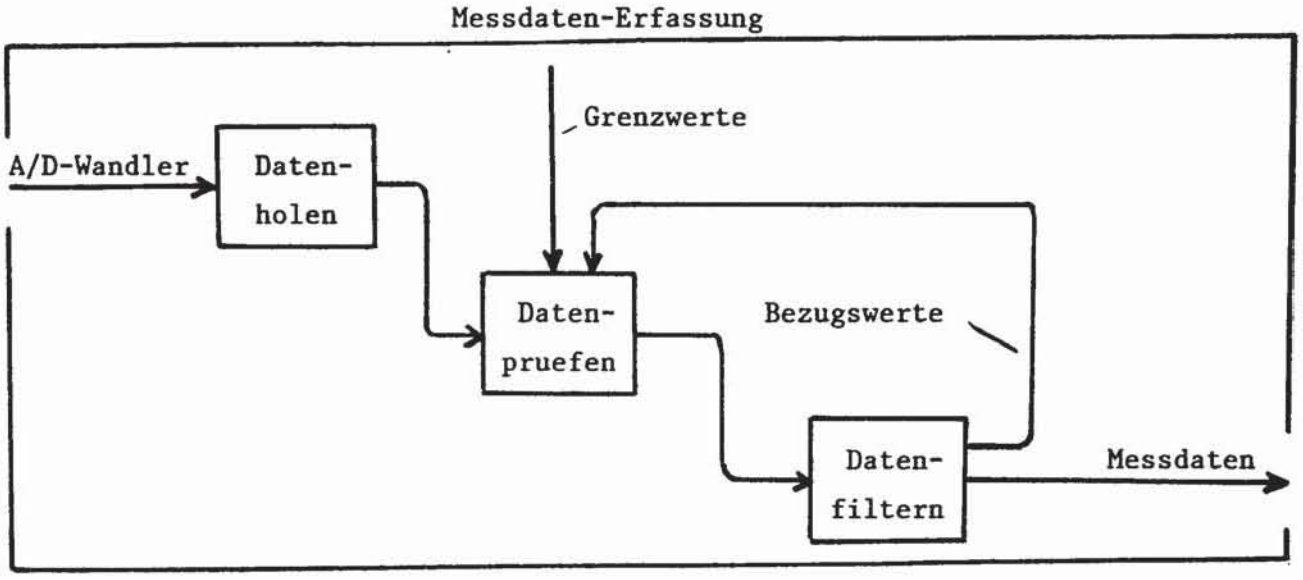
Beispiel: In PSL (Teichroew, Hershey, 1977) lassen sich die wichtigsten Elemente der Sprache (Objekte der Art PROCESS und andere) rekursiv zerlegen (also: PROCESS a besteht aus den PROCESSES x, y, z). Das Werkzeug PSA stellt bereits bei der Eingabe sicher, dass die entstehende Struktur baumartig ist (PROCESS y darf also nicht gleichzeitig Bestandteil von PROCESS b sein).

```
PROCESS Messdatenerfassung;
   DESCRIPTION;
      greift direkt auf die AD-Wandler zu, liefert gefilterte Daten;
   SUBPARTS ARE Daten-holen, Daten-pruefen, Daten-filtern;   /* Verfeinerung */

PROCESS Daten-pruefen;                          /* wird nun weiter verfeinert */
   SUBPARTS ARE Grenzwert-Pruefung, Differenz-Pruefung;

EOF                                             /* beendet die PSL-Spezifikation */
```

3.1.2 Datenfluss-orientierte Modelle

Gerade bei leittechnischen Systemen ist der Datenfluss ein wesentlicher Aspekt. Schon die primitivste Vorstellung ("Was geht rein, was kommt raus ?") beruht auf einem Datenfluss-orientierten Verständnis. Daher ist die Beschreibung von Datenflüssen eines der wichtigsten Spezifikationskonzepte.

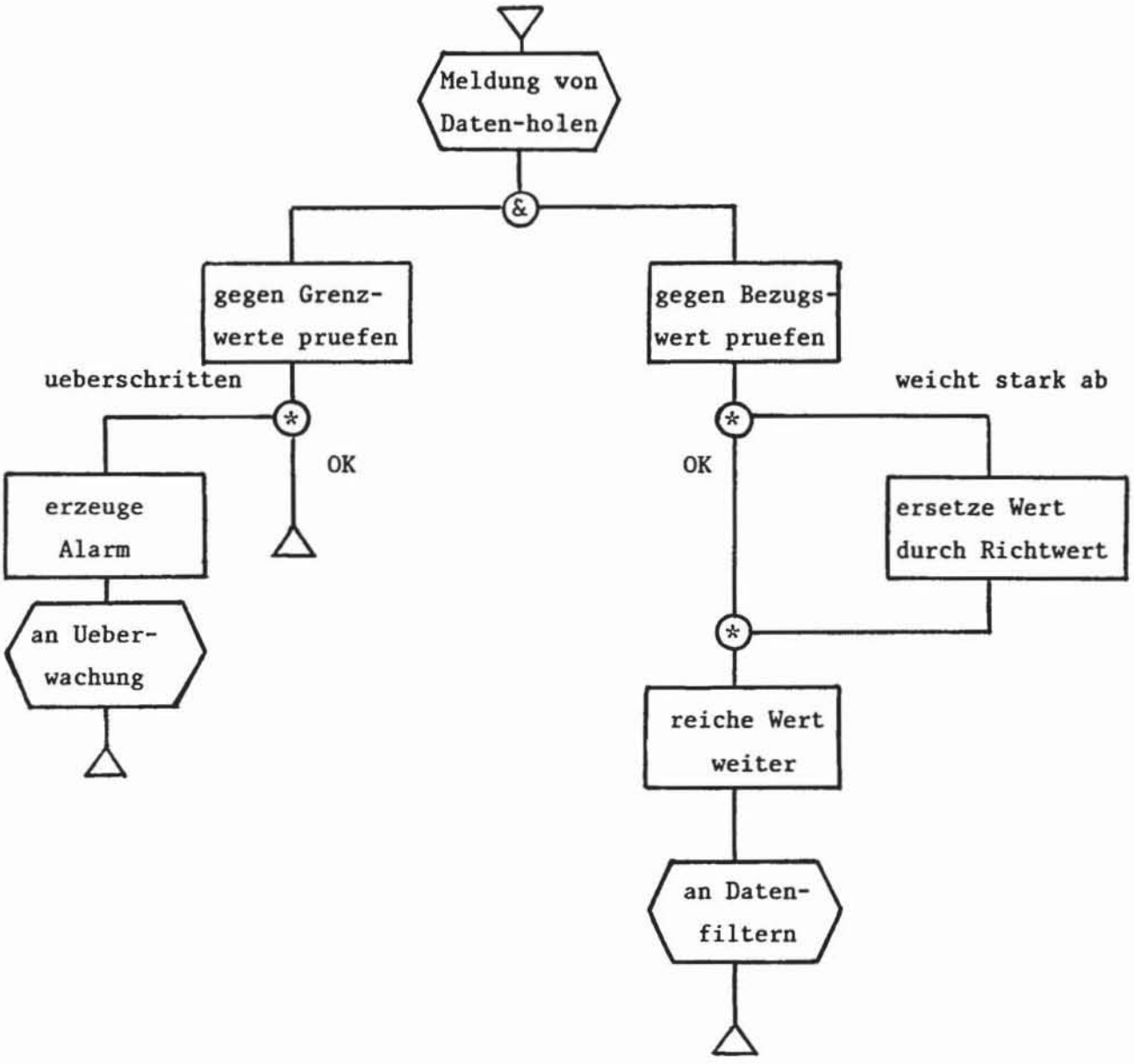
Beispiel: In SADT (Ross, Schoman, 1977) wird das System durch Actigrams und Datagrams beschrieben. In den Actigrams erscheinen die Aktivitäten als Kästchen, die Datenflüsse als Verbindungslinien. In den Datagrams ist das Verhältnis umgekehrt, die Kästchen sind Daten, die Verbindungen Operationen auf den Daten. Die Praxis zeigt, dass die Actigrams wesentlich leichter zu zeichnen sind (vereinfachtes Bild, vgl. Racke, Rombach, 1983, in diesem Band).



3.1.3 Ereignisorientierte Modelle

Die wesentliche Funktion vieler leittechnischer Systeme ist die Reaktion ("response") auf ein Ereignis ("stimulus") in ihrer Umgebung. Daher basieren viele Modelle auf dem stimulus-response-Konzept, das eng verwandt ist mit dem (mathematischen) Automaten-Begriff (Vitins, 1983). Besonders dann, wenn wir von der Arbeit des leittechnischen Systems die Vorstellung haben, dass es auf stochastische Anforderungen aus der Umgebung hin innerhalb einer begrenzten Zeit bestimmte Aktionen ausführen muss, ist dieses Prinzip angemessen.

Beispiel: Den sogenannten R_NETS (Requirement-Nets) des Spezifikationssystems SREM liegt das Stimulus-Response-Konzept zugrunde. Jedes R_NET beginnt oben mit einem auslösenden Ereignis und endet mit einer Reaktion.



3.1.4 Axiomatische Modelle und Abstrakte Datentypen

Jede Sprache, die sich nicht auf Begriffe abstützen soll, die als allgemein bekannt vorausgesetzt werden, benötigt Axiome. So werden etwa die Begriffe der Mathematik axiomatisch eingeführt. Nur so ist es im strengen Sinne möglich, Beweise zu führen.

Das Prinzip der axiomatischen Definition wird auch für Spezifikationen angewendet. Es ermöglicht hier nicht nur die Programm-Verifikation, sondern unterstützt auch das Bestreben, in der Spezifikation möglichst auf Aussagen über die Realisierung zu verzichten. Da ein komplexes Software-System nicht durch eine einzige unstrukturierte Menge von Axiomen fassbar ist, werden in der Praxis zunächst Objekte als Abstrakte Datentypen separiert und dann einzeln axiomatisch, d.h. ohne Bezugnahme auf die innere Struktur der Realisierung, beschrieben. Dieser Ansatz wurde von Parnas (1972) eingeführt.
Axiomatische Spezifikationen gelten als recht schwer schreib- und lesbar.

Beispiel: Die Sprache SPECIAL (Levitt, Robinson, Silverberg, 1980) beruht auf dem oben skizzierten Prinzip. Daher besteht für SPECIAL auch ein automatisches Verifikationssystem.

Ein einigermassen sinnvolles Beispiel würde den Rahmen dieses Ueberblicks sprengen. Hier sei nur kurz angedeutet, dass die sogenannten O-functions den Zustand eines Datentyps verändern, während die V-functions den Zustand abzufragen gestatten. Das Prinzip besteht nun darin, die Wirkung der O-functions durch die Werte der V-functions vorher und nachher (unterschieden durch vorgesetztes Hochkomma) zu definieren. Der folgende Ausschnitt einer SPECIAL-Spezifikation (Chandersekaran, Linger, 1981) soll nur einen optischen Eindruck vermitteln. Der Modul enthält viel mehr Definitionen als hier gezeigt.

```
MODULE file_system

VFUN get_file_pointer (file f) [fileset fs] -> INTEGER i;
  EXCEPTIONS    Error1: no_file_exists (fs, f);
  INITIALLY     i = ?;

OFUN write_file (file f; datablock db) [INTEGER block; fileset fs];
  EXCEPTIONS    Error1: no_file_exists (fs, f);
  EFFECTS       'read_block (f, get_file_pointer (f, fs)) = db;
                'get_file_pointer (f, fs) = get_file_pointer (f, fs) + 1;
                IF get_file_pointer (f, fs) = get_current_length (f) + 1
                THEN 'get_current_length (f) = get_current_length (f) + 1
                ELSE 'get_current_length (f) = get_current_length (f);
END_MODULE.
```

3.1.5 Prädikaten-Logik

Die Prädikaten-Logik ist ein mächtiges Kalkül, das die Formalisierung aller Aussagen gestattet, denen sich die Beurteilung "wahr" oder "falsch" zuordnen lässt. Dieses wichtige Modell der Mathematiker wurde von Seiten der Artificial Intelligence auch in der Informatik eingeführt.

Beispiel: Das PROLOG-System (Clocksin, Mellish, 1981), das in unterschiedlichen Realisierungen in vielen Ländern der Welt existiert, bietet die für die Arbeit im Prädikaten-Kalkül notwendige Unterstützung. Objektmengen können durch logische Attribute beschrieben und miteinander in Beziehung gesetzt werden. Durch Operationen auf die Mengen lassen sich Objekte mit bestimmten Eigenschaften oder Aggregate aus verbundenen Objekten ermitteln. Auch hier kann das Beispiel nur einen sehr oberflächlichen Eindruck der Mächtigkeit geben.

```
umfasst (messdaten-erfassen, daten-holen, daten-pruefen, daten-filern).
umfasst (daten-pruefen, grenzen-pruefen, abweichung-pruefen).
verwendet (daten-holen, ad-wandler).
erzeugt   (daten-holen, rohdaten).
verwendet (daten-pruefen, rohdaten).
erzeugt   (daten-pruefen, gepruefte-daten).
verwendet (daten-filtern, gepruefte-daten).
erzeugt   (daten-filtern, messdaten).
```

```
/* "umfasst", "verwendet", "erzeugt" sind einfache Prädikate über einer      */
/* Objektmenge. Daraus lassen sich kompliziertere aufbauen, etwa so:         */

untergeordnet (X, Y):- umfasst (Y, X).
untergeordnet (X, Y):- umfasst (Y, Z), untergeordnet (X, Z).     /* rekursiv */

/* Danach lassen sich z.B. folgende Frage an das PROLOG-System stellen :     */

?- erzeugt (daten-holen, WAS), verwendet (daten-pruefen, WAS).
WAS = rohdaten.                                          /* Antwort von PROLOG */
?- erzeugt (daten-holen, AUCH), verwendet (daten-filtern, AUCH).
no                                                       /* Antwort von PROLOG */
?- untergeordnet (grenzen-pruefen, messdaten-erfassen).
yes                                                      /* Antwort von PROLOG */
```

3.2 Methoden

Alle Methoden für die Spezifikation gehen von der bekannten, aber unpopulären Tatsache aus, dass jede Information, die irgendeine Bedeutung im Projekt hat, festgehalten werden und später zugänglich sein muss. Die Information sollte auch so früh wie möglich formalisiert werden, damit sie durch automatische Werkzeuge bearbeitet werden kann.

Eine weitere Tatsache ist, dass die Beschreibung eines grossen Systems viel Zeit benötigt. Daher muss die Methode ein schrittweises Vorgehen unterstützen. Bei der Programmierung geschieht dies durch Strukturierung (Verwendung von Prozeduren), die sich im allgemeinen an hierarchischen Beziehungen orientiert ("program development by stepwise refinement", Wirth, 1971). Für Spezifikationssysteme ist dieses Verfahren zu restriktiv; der Anwender soll die Möglichkeit haben, Informationen in praktisch beliebiger Reihenfolge zu sammeln und - auch schon im unvollständigen Zustand - zu überprüfen und auszuwerten. Da all dies durch das Spezifikationssystem unterstützt werden soll, müssen Sprachen und Werkzeuge entsprechend konstruiert sein.

Die Methode sollte den Anwender dabei unterstützen, seine Anforderungen möglichst vollständig zu erfassen. Ein Weg dahin ist, den Prozess der Erfassung selbst zu strukturieren, z.B. dadurch, dass nicht "einfach" alle Anforderungen aufgezählt (oder eben vielfach: vergessen) werden, sondern dass nacheinander zu verschiedenen Aspekten Stellung bezogen wird. So wird beispielsweise das gewüschte Verhalten des leittechnischen Systems in Fehlersituationen wahrscheinlich nicht übersehen, wenn ausdrücklich danach gefragt ist. Alternativen solcher Aspekt-Gruppen sind

- Trennung verschiedener Strukturen
  (z.B. hierarchischer Aufbau und Datenflüsse),
- Einordnung in gewissen Anforderungskategorien
  (z.B. in Funktions- und Leistungsanforderungen),
- Zuordnung zu gewissen Problemkomplexen
  (z.B. Normal- oder Notfallverhalten).

Die Trennung verschiedener Aspekte kann mit der Verantwortung unterschiedlicher Personengruppen verknüpft werden.

3.3 Sprachen

Nachdem in 3.1 bereits die Modelle besprochen wurden, geht es hier nur noch um den syntaktischen Teil der Sprachen, also um die Art und Weise, wie eine bestimmte Information dargestellt wird.

3.3.1 Lineare und zweidimensionale Sprachen

Der augenfälligste Unterschied besteht zwischen solchen Sprachen, die nur Text verwenden (lineare Sprachen), und graphischen Formen (zweidimensionale oder graphische Sprachen). Es ist wichtig zu beachten, dass der Unterschied grundsätzlich nur in der Repräsentation liegt: Eine Baumstruktur beispielsweise kann sowohl linear als auch graphisch dargestellt werden. Allerdings fällt es den meisten Menschen leichter, graphische Information aufzunehmen. Ausserdem ist die Umsetzung in Graphik ein wichtiger Prüfstein für formale Sprachen: Ist sie nur schwer möglich, so überfordert die Sprache vermutlich auch die Vorstellungskraft der Benutzer.

In der Praxis sind die graphischen Sprachen niemals frei von Text, und lineare Sprachen sind nur lesbar, wenn ein Minimum an graphischer Strukur vorhanden ist (ein Befehl pro Zeile, Einrückungen, Leerzeilen usw.).

Bisher waren die üblichen Peripheriegeräte (zeichenorientierte Bildschirme, Tastaturen, Schnelldrucker) und die mangelnde Portabilität graphischer Software ein starker Hinderungsgrund für den Einsatz zweidimensionaler Sprachen. Durch neue Geräte (hochauflösende Bildschirme, "Maus" oder andere zweidimensionale Eingabe, Laserprinter) und durch Standardisierung der Graphiksoftware (GKS, siehe Pfaff, 1982) zeichnet sich gegenwärtig eine Aenderung ab.

3.3.2 Prozedurale und nicht prozedurale Sprachen

Gängige Programmiersprachen sind prozedural: Die Befehle werden in der Reihenfolge ausgeführt, in der sie aufgeschrieben sind. Abweichungen müssen vom Programmierer explizit angegeben werden (Sprunganweisungen).

Für die Spezifikation hat dieses Prinzip Nachteile: Zum einen erhält die Reihenfolge der Informationen eine Bedeutung, die sie im Interesse der ungeordneten Erfassung (siehe 3.2) nicht haben sollte. Zum andern werden Ausführungsreihenfolgen festgelegt, wo diese Festlegung nicht (oder noch nicht) erforderlich ist. Dies widerspricht dem Prinzip, die Freiheit der Realisierung nicht unnötig einzuengen. Daher sind Spezifikationssprachen in der Regel nicht-prozedural, d.h. die Reihenfolge der Aufschreibung bestimmt nicht die Reihenfolge der Ausführung (Leavenworth, Sammet, 1974).

Beispiel: Die Beschreibung eines Objekts der Art "procedure" kann in ESPRESO (Ludewig, 1982b) etwa wie folgt aussehen:

```
procedure Vorcheck-der-Messwerte:
    reads A-D-Wandler;
    produces Rohwerte;
end Vorcheck-der-Messwerte.
```

Ueber die Reihenfolge der durch reads und produces beschriebenen Aktionen ist dadurch nichts ausgesagt.

3.4 Werkzeuge

Rein physisch begegnen dem Anwender Werkzeuge in Form von Hardware, etwa als Bildschirm-Terminal oder als Drucker. In dieser Hinsicht hat man sich bisher meist mit den Standardgeräten eingerichtet und keine speziellen Anforderungen gestellt (vgl. aber 3.3.1). Mit dem Wort "Werkzeug" bezeichnet man daher die Programme, mit deren Hilfe Spezifikationen bearbeitet werden können: Programme für die Eingabe und Speicherung, für die Wiedergabe, Auswertung und Prüfung und für die Umsetzung in andere Darstellungsformen.

3.4.1 Eingabe und Speicherung

Der wichtigste Vorteil, der sich bei der rechnerunterstützten Spezifikation bietet, ist die problemlose Speicherung grosser Informationsmengen. Diese Speicherung kann primitiv mit Hilfe von Dateien stattfinden oder - was sich immer mehr durchsetzt - in einem Datenbank-System. Pionier war dabei, wie auch in manch anderer Hinsicht, das ISDOS-Projekt mit PSA. Die Datenbank hat den Vorteil, durch logische Anordnung der Information die Suche nach bestimmten Objekten oder Angaben und damit die Realisierung aller anderen Werkzeuge wesentlich zu erleichtern.

Wenn die Spezifikationen nicht einfach in Dateien gespeichert werden, ist bei der Eingabe eine syntaktische Analyse erforderlich. Diese kann unterschiedlich weit gehen: Im einen Extremfall deckt sie nur die lexikalische Ebene ab (PASILA, Balzert, 1981), so dass beispielsweise die inkonsistente Verwendung von Namen nicht beanstandet wird, im andern reicht sie bis in die Semantik der Angaben hinein (PSL/PSA, ESPRESO, siehe Beispiel bei 3.1.1). Damit erhält der Benutzer eine Kontrolle seiner Angaben, und der Datenbankinhalt bleibt stets konsistent.

3.4.2 Wiedergabe, Auswertung und Prüfung

Ist die Spezifikation in einer nicht unmittelbar lesbaren Form gespeichert, so muss der Benutzer mit Hilfe des Werkzeugs zugreifen. In den meisten Fällen will er nicht die gesamte Spezifikation sehen, sondern nur bestimmte Teile daraus, etwa alle Angaben zu einem speziellen Objekt oder alle Information, die einem bestimmten, formal definierten Aspekt zugeordnet ist. Wie ein allgemeines Datenbank-System muss das Spezifikationswerkzeug also Retrieval-Funktionen anbieten.

Je nachdem, welche Informationen in der Spezifikation vorgesehen sind, lassen sich gewisse Auswertungen automatisch vornehmen. Wird etwa bei jedem spezifizierten Element angegeben, wie hoch der Realisierungsaufwand geschätzt wird, so kann das Werkzeug daraus automatisch die Summe bilden. Ebenso kann es Informationen darüber liefern, welche Objekte bereits erwähnt wurden, ohne explizit spezifiziert zu sein.

Darüberhinaus lassen sich Informationen auch auf bestimmte formale Eigenschaften prüfen. So sollte z.B. jedes im System erzeugte Ergebnis an irgendeiner Stelle verwendet werden. Andernfalls ist die Spezifikation vermutlich unvollständig. Solche Prüfungen können vom Werkzeug automatisch vorgenommen werden.

3.4.3 Umsetzung in andere Darstellungsformen

In 2.2 wurde gesagt, dass die Syntax nur eine Hülle der Information darstellt. Das Werkzeug erlaubt es, automatisch zwischen verschiedenen Darstellungen zu wechseln. Vor allem die Erzeugung graphischer Darstellungen aus der Spezifikation ist sehr beliebt (ein wichtiges Element in EPOS, Biewald et al., 1979). Ebenso lassen sich Tabellen drucken.

Mit Hilfe von Werkzeugen is es auch möglich, zwischen unterschiedlichen Gliederungsprinzipien zu wechseln: So können etwa in PSL alle Objekte eines bestimmten Typs in einem Zuge spezifiziert werden. Ist die Spezifikation einmal in der Datenbank, so kann das Werkzeug PSA z.B. den Datenfluss ausgeben oder die hierarchische Struktur.

4. Heutiger Stand und Trends

4.1 Stand der Technik

PSL/PSA ist schon seit Jahren in vielen, vorwiegend amerikanischen Firmen im Einsatz. Zu den Anwendern gehören beispielsweise Bell Telephone, Hughes Aircraft und Chase Manhattan Bank. Am Beispiel von ISDOS kann man sehen, wie ein Spezifikationssystem erfolgreich vertrieben werden kann: Die Anwender erhalten das System zur Miete (ca. 15 000 $/Jahr). Durch einen regelmässig erscheinenden Rundbrief werden laufend Neuerungen mitgeteilt und Veranstaltungen für die Mitarbeiter der Sponsor-Firmen (Information, Schulung, Erfahrungsaustausch) verbreitet.

Anscheinend haben viele Firmen für ihre speziellen Anwendungen Spezifikationssysteme entwickelt (oder vorhandene adaptiert), die nicht für den Software-Markt bestimmt sind und daher der Oeffentlichkeit meist unbekannt bleiben (siehe z.B. Taylor, 1980).

Systematische Verfahren werden heute in vielen Firmen angewendet. Diese Verfahren sind beruhen meist auf Vorbildern wie SADT oder der Jackson-Methode. Die Implementierung spezieller Werkzeuge ist nur grossen Anwendern möglich, daher werden oft vorhandene Mittel wie Editoren und File-Systeme eingesetzt und nur in Sonderfällen ergänzt.

Allgemein sind die Werkzeuge noch immer der Flaschenhals. Der Implementierungsaufwand ist recht hoch (je nach Komfort zwischen 5 und 50 Mannjahren oder mehr) und der Erfolg ungewiss. Ausserdem verfügen nur wenige Firmen über das notwendige Know-How. Die meisten der auf dem Markt angebotenen Werkzeuge sind im Katalog von Reifer (o.J.) zu finden.

Eine Entspannung des Konflikts zwischen Aufwand und Risiko wird durch den Einsatz von Meta-Systemen erreicht. Bei diesen wird die eigentliche Spezifikationssprache nicht in das Werkzeug eingebaut, sondern formal definiert. Das Meta-System generiert daraus das Spezifikationssystem. Auf diese Weise kann das Spezifikationssystem neuen Erkenntnissen oder geänderten Anforderungen mit geringem Aufwand angepasst werden (z.B. PASILA, Balzert, 1981). Auch hier war ISDOS der Vorreiter.

Weit fortgeschritten ist der Einsatz von Spezifikationssystemen dort, wo der Gesamtaufwand und das Risiko fehlerhafter Software sehr hoch sind, also bei militärischen Anwendungen und in der Kerntechnik. Viele Systeme sind speziell für diese Anwendungen geschaffen worden, z.B. SREM (Alford, 1977, 1980) und HOS (Hamilton, Zeldin, 1976).

4.2 Stand der Wissenschaft

In der wissenschaftlichen Arbeit sind folgende Richtungen zu unterscheiden:

- Die axiomatische Spezifikation wird durch komfortablere Sprachen leichter anwendbar und rückt damit langsam in die Nähe der Praxis. Das gleiche gilt für PROLOG. Weisse Flecken, etwa die Behandlung der Paralllelität, werden weiterhin erforscht.

- Wenn eine Spezifikation voll formalisiert ist, sollte sie bei den weiteren Bearbeitungen vor Fehlern geschützt sein. Dies lässt sich dadurch erreichen, dass der Code aus der Spezifikation durch eine Reihe formaler Transformationen erzeugt wird. Dazu ist die Intuition des Designers erforderlich. Ein Werkzeug kann aber die Transformationen durchführen und damit kontrollieren. Dieses Prinzip wird in CIP eingesetzt (Bauer, 1979).

- Arbeiten auf den Gebieten der Datenbanken und der Künstlichen Intelligenz liefern ebenfalls Beiträge zur Spezifikation.

Noch immer ungeklärt ist die Frage, wie der technische Prozess, um den es ja schliesslich beim Einsatz des Prozessrechensystems geht, formal beschrieben werden kann. Eine dazu geeignete Sprache müsste sowohl am Problem, d.h. an der Denkwelt des Ingenieurs, als auch an den Erfordernissen einer formalisierten Spezifikation orientiert sein.

4.3 Trends

Die erste Euphorie, in der man die Spezifikation als Schlüssel zu allen Software-Problemen behandelt hat, ist in den letzten Jahren gedämpft worden. Dies hatte vor allem folgende Ursachen:

- Ein universelles Basismodell ist noch immer nicht in Sicht. Es scheint, dass in Zukunft eher Systeme für spezielle Anwendungen kommen werden. Aber auch bei diesen wurde bisher kein befriedigender Kompromiss zwischen Verständlichkeit (vor allem für den Kunden) und Exaktheit (für den Implementierer) gefunden. Einzig dort, wo der Kunde selbst an formale Beschreibungen gewöhnt ist (etwa in der Telephon-Technik), lässt sich diese Kluft überbrücken.

- Wenn endlich eine Spezifikation vorliegt, erhebt sich die Frage, wie daraus der Code entwickelt werden kann. Dieser Schritt wird von den gängigen Systemen kaum unterstützt. Auch ist es mühsam (und wird praktisch meist vernachlässigt), die Spezifikation später nachzuführen.

- Im Gegensatz zu Spielprogrammen, wie sie im universitären Bereich meist betrachtet werden, entsteht industrielle Software nicht "auf der grünen Wiese", sondern durch Evolution aus vorhandenen Systemen. Dazu bieten die Spezifikationssysteme kaum Unterstützung.

- Der Life-Cycle in seiner üblichen Form verliert, wie oben erwähnt wurde, an Bedeutung. Vor allem das Prinzip des Fast Prototyping wird jetzt sehr stark propagiert. Dieses Vorgehen ist in den heutigen Spezifikationssystemen nicht berücksichtigt.

- Moderne Sprachen wie Ada (DoD, 1980) bieten Möglichkeiten, die bisher nur in Spezifikationssprachen vorhanden waren (z.B. für die Spezifikation von Schnittstellen).

Die genannten Gründe legen es nahe, den gesamten Prozess der Software-Entwicklung im Zusammenhang zu sehen. An die Stelle einzelner Hilfsmittel, die logisch unabhängig voneinander sind, tritt damit das sogenannte Software Engineering Environment (Hünke, 1981; Howden, 1982). Natürlich wird in einem solchen Environment das Spezifikationssystem eine wichtige Komponente sein.

Trägt man den Formalisierungsgrad über den Stationen der Software-Entstehung auf (Abbildung 3, nach Ludewig, Streng, 1978), so bilden die Spezifikationssysteme darin einen Trittstein, wie man ihn benutzt, um einen Bach zu überqueren. Environments sollen stattdessen in Zukunft eine stabile Brücke bilden.

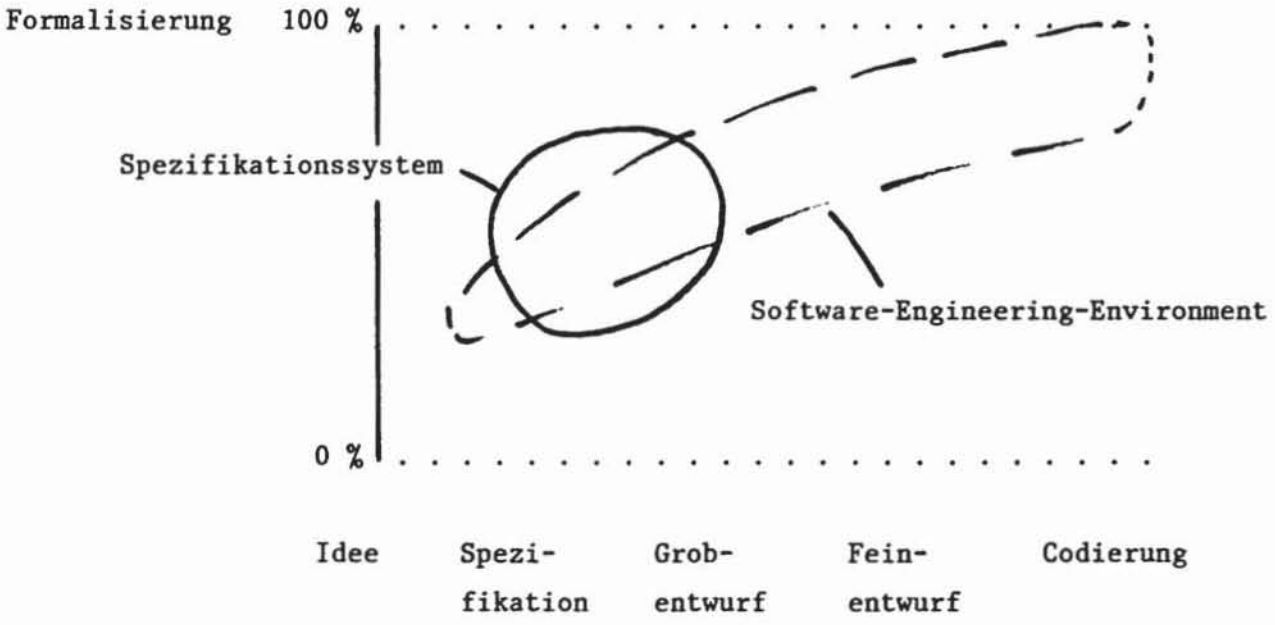


Abb. 3: Formalisierungsgrad über Software-Entwicklungsphase

Bei BBC (Abteilung NLS) hat man mit der Entwicklung und Einführung des PDE (Program Development Environment) einen praktischen Schritt zum Software Engineering Envoironment getan. PDE unterstützt vorerst im wesentlichen die Codierung, soll jedoch in alle Richtungen erweitert werden. In dieselbe Richtung zielt die Arbeit des Projekts Software-Technik der Gruppe Informatik im Brown Boveri Forschungszentrum. Auch hier arbeiten wir an Beiträgen für den zukünftigen "Software Arbeitsplatz".

5. Erfahrungsregeln für den Einstieg in Spezifikationssysteme

Aufgrund der Erfahrungen lassen sich einige Feststellungen treffen, die nicht vom speziellen Spezifikationssystem abhängen. Diese Regeln sind:

- Die Umstellung von der traditionellen auf eine formalisierte, durch ein Werkzeug unterstützte Methode der Spezifikation ist schwierig. Der Schritt ist wesentlich grösser als der Uebergang von einer primitiven auf eine höhere Programmiersprache. Entsprechend muss der Aufwand für Schulung und Einarbeitung veranschlagt werden.

- Programmiererfahrung ist keineswegs vorteilhaft. Nicht-Programmierer haben geringere Schwierigkeiten bei der abstrakten Spezifikation, weil sie nicht sofort an die Realisierung und Optimierung denken (D. Teichroew). Programmierer neigen dazu, die Konzepte der Spezifikationssprache zur Codierung zu missbrauchen.

- Ein Software-Entwicklungssystem kann das Management unterstützen, nicht ersetzen. Ein starkes Management ist auch erforderlich, um die rasche Einführung sicherzustellen.

- Der Einstieg in ein Spezifikationssystem ist eine mittelfristige Investition, keine Feuerwehrübung. Er sollte keinesfalls bei einem Projekt versucht werden, dessen Scheitern sich bereits abzeichnet. Im Gegenteil ist es vorteilhaft, ein Pilotprojekt zu wählen, dessen Termine einen gewissen Bewegungsraum lassen und in dem gute, womöglich begeisterte Leute arbeiten. Sie werden am leichtesten die Anfangsschwierigkeiten überwinden und durch ihre Mund-zu-Mund-Propaganga die Akzeptanz bei den übrigen Mitarbeitern verbessern. Mindestens ein Fachmann muss sich auf die Einführung selbst konzentrieren, die Benutzer beraten und die Erfahrungen sammeln und auswerten (vgl. Reinshagen, 1983, in diesem Tagungsband).

- Die wichtigste Wirkung eines Spezifikationsssystems ist katalytischer Natur: Die Kommunikation wird leichter, die Dokumentation besser, die Qualität höher, und zwar primär nicht durch die Hilfe eines Werkzeugs, sondern durch den Druck einer einheitlichen, wohldefinierten Vorgehensweise. Grundsätzlich lässt sich dieser Fortschritt auch ohne ein Spezifikationssystem erreichen, nur bedarf es dazu einer starken Disziplin. Das von aussen kommende, "neutrale" Spezifikationssystem hat es leichter als der starke Mann in der DV-Abteilung.

6. Literaturverzeichnis

Für ein ausführlicheres Literaturverzeichnis vergleiche den Aufsatz von Racke und Rombach in diesem Tagungsband.

Alford, M. (1977):
A requirements engineering methodology for real time processing requirements.
IEEE Transactions on Software Engineering, SE-3, 60-69.

Alford, M. (1980):
Software requirements engineering methodology (SREM) at the age of four.
COMPSAC 80, Chicago, ???.

Balzert, H. (1981):
Methoden, Sprachen und Werkzeuge zur Definition, Dokumentation und Analyse von Anforderungen an Software-Produkte.
GI-Informatik-Spektrum, 4, 3, 145-163 und 4, 246-260.

Bauer, F.L. (1972):
Software Engineering.
in Freiman, C.V.: Proc. of the IFIP Congress 71, North Holland Publishing Company, Amsterdam, New York, Oxford, pp.530-538;
nachgedruckt in Bauer, F.L. (ed.) (1973): Software Engineering.
Lecture Notes in Computer Science, Vol.30,
Springer-Verlag, Berlin, Heidelberg, New York, pp. 522-545.

Bauer, F.L. (1979):
Program development by stepwise transformation - the project CIP.
in Bauer, Broy (ed.): Program construction.
Lecture Notes in Computer Science, Vol.69,
Springer Verlag, Berlin, Heidelberg, New York, 237-266.

Biewald, J., P. Göhner, R. Lauber, H. Schelling (1979):
EPOS - a specification and design technique for computer controlled real-time automation systems.
4th Intern. Conf. on Software Engineering, München, 1979,
IEEE Cat. No. 79 CH 1479 - 9C, pp.245-250.

Boehm, B.W. (1976):
Software Engineering.
IEEE Transactions on Computers, C-25, pp.1226-1241.

Chandersekaran, C.S., R.C. Linger (1981):
Software specification using the SPECIAL language.
The Journal of Systems and Software, 2, 31-38.

Clocksin, W.F., C.S. Mellish (1981):
Programming in PROLOG.
Springer Verlag, Berlin, Heidelberg, New York.

DoD (1980):
Reference Manual for the ADA programming language.
United States Department of Defense. Zahlreiche Nachdrucke.

Goos, G. (1973):
Hierarchies.
in Bauer, F.L. (ed.): Software Engineering.
Lecture Notes in Computer Science, Vol.30, Springer, pp.29-46.

Hamilton, M., S. Zeldin (1976):
Higher Order Software - a methodology for defining software.
IEEE Transactions on Software Engineering, SE-2, 9-32.

Hice, G.F., W.S. Turner, L.F. Cashwell (1974):
System development methodology.
American Elsevier, New York.

Howden, W. (1982):
Contemporary software development environments.
Comm. ACM, 25, 5, 318-329.

Hünke, H. (ed.) (1981):
Software Engineering environments.
Proc. of the Symposium held at Lahnstein, June 16 - 20, 1980.
North Holland Publishing Company, Amsterdam, New York, Oxford.

Kramer, J. (ed.) (1982):
Glossary of terms.
EWICS TC on Application Oriented Specification.
erhältlich von Jeffrey Kramer, Imperial College, Dept. of Computing,
180 Queen's Gate, GB - London SW7 2BZ.

Leavenworth, B.M., J.E. Sammet (1974):
An overview of nonprocedural languages.
SIGPLAN Notices, 9, No.4, 1-12.

Levitt, K.N., L. Robinson, B.A. Silverberg (1980):
Writing simulatable specifications in SPECIAL.
in Berg, Giloi (Hrsg.): The use of formal specifications of software.
Informatik Fachbericht Nr. 36, Springer-Verlag, Berlin, Heidelberg,
New York, pp. 39-78.

Ludewig, J., W. Streng (1978):
Methods and tools for software specification and design - a survey.
EWICS TC on Safety and Security, Paper No. 149, 23 Seiten.

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

Ludewig, J. (1982b):
ESPRESO - A system for process control software specification.
7th Conf. on operating systems, Visegrad, Ungarn, Januar 1982.
Erscheint in IEEE Transactions on Software Engineering, ca. Juli 1983.

McCracken, D., M. Jackson (1982):
Life cycle concept considered harmful.
ACM Software Engineering Notes, 7, 2, 29-32.

Parnas, D.L. (1972):
A technique for software module specification with examples.
Commun. ACM, 15, 330-336.

Parnas, D.L. (1974):
On a 'buzzword': hierarchical structure.
in J.L. Rosenfeld (ed.), Information Processing 74,
North Holland Publishing Company, Amsterdam, New York, Oxford, pp.336-339.

Pfaff, G. (1982):
Die Konzeption, Programmierung und Anwendung des Graphischen Kernsystems GKS.
in J. Nehmer (Hrsg.), GI - 12. Jahrestagung, Informatik-Fachberichte
Band 57, Springer-Verlag, Berlin, Heidelberg, New York, pp.152-169.

Reifer Consultants (o.J.):
Software Tools Directory.
Loseblattsammlung (wird immer wieder ergänzt), Reifer Consultants, Inc.,
2733 Pacific Coast Highway, Suite 203, Torrance, California 90505, USA.

Ross, D.T., K.E. Schoman (1977):
Structured analysis for requirements definition.
IEEE Transactions on Software Engineering, SE-3, 6-15.

Simon, H.A. (1962):
The architecture of complexity.
Proc. of the American Philosophical Society, 106, 467-482.

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

Taylor, B. (1980):
A method for expressing the functional requirements of real-time
systems.
in Haase (ed.): IFAC/IFIP Workshop on real-time programming.
Pergamon Press, Oxford etc., 111-120.

Teichroew, D., H. Sayani (1971):
Automation of system building.
DATAMATION, 17, No.8, 25-30.

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.

Vitins, M. (1983):
On the specification of system behaviour.
Brown Boveri Research Center, Internal Report.

Wirth, N. (1971):
Program development by stepwise refinement.
Commun. ACM, 14, 221-227.