COMPASS - Portable Werkzeuge für den Entwurf und die Abwicklung von Software - Projekten

P. Winkler, AEG Software-Technik

Zusammenfassung

COMPASS-Werkzeuge gehoeren zum modernen Software-Engineering.

COMPASS ist ein Software-paket, das durch uebersichtliche, bildhafte und kommunikationsfoerdernde Mittel die strukturierte Darstellung in allen Projektphasen wirkungsvoll unterstuetzt.

COMPASS wird auch zur rationellen Entwicklung und Produktion von Software fuer alle Phasen des Software-Life-Cycles eingesetzt.

COMPASS erleichtert die vollstaendige Beschreibung in jeder Projektphase, verbindet dauerhaft Entwurf und Code und bietet eine schritthaltende aktuelle Dokumentation.

Summary

COMPASS Tools are parts of modern software-engineering.

COMPASS is a software package which efficiently supports the structured representation of all phases of a project by using clear, graphical and communication encouraging means.

COMPASS is also used for the efficient develoment and production of software for all phases of the software-life-cycle.

COMPASS makes the complete description in each phase of the project easy. It permanently combines design and specification with the code and offers an up-to-date documention.

1. Uebersicht

Die COMPASS-Werkzeuge wollen den Methodenstreit nicht um eine weitere Variante bereichern, sondern den Software-Ingenieur bei der Loesung von Automatisierungsvorhaben wirkungsvoll unterstuetzen.
Er kann sich verschiedener gaengiger Methoden bedienen:
Funktionale Verfeinerung,
Datenstromorientierter Entwurf,
Datenstrukturierter Entwurf.

Im folgenden wird das COMPASS-System anhand eines Beispiels, das mit PEARL programmiert wurde, dargestellt.

Das COMPASS-System bietet Werkzeuge an zur
- Dokumentation
- graphikorientierten Darstellung der Aufgabenstellung, der Spezifikation und des Feinentwurfes
- automatischen Pseudocode-Generierung
- Unterstuetzung der Codierung und Compilierung
- Unterstuetzung bei Test und Wartung

Bei jedem Software-Projekt ist die schritthaltende Dokumentation auf jeder Bearbeitungsebene eine der wichtigsten Aufgaben. Die COMPASS-Werkzeuge bieten Unterstuetzung bei der rationellen Entwicklung und Produktion von Software in allen Phasen des Software-"life-cycle".
Auch beim "rapid prototyping", das jedoch eine genuegend grosse Anzahl von wiederverwendbaren Software-Moduln voraussetzt, sind die COMPASS- Werkzeuge von besonderem Nutzen. Unzureichende Dokumentation, fehlende Modularitaet und Strukturierung verhinderten bisher allzu oft die Gewinnung von wiederverwendbarer Software aus abgewickelten Software-Projekten. Diese Luecke kann man durch den Einsatz von unterstuetzenden Werkzeugen verkleinern.

Die Dokumentation eines Software-Produktes darf aber nicht nur ein losgeloester Zweig der Softwareproduktion sein, sondern muss mit dem Entwurf und dem Code so integriert sein, dass bei Aenderungen und Ergaenzungen zwischen der Dokumentation und den ausgefuehrten Programmen keine Differenzen entstehen.

Die COMPASS-Werkzeuge unterstuetzen durch uebersichtliche graphikorientierte und damit kommunikationsfoerdernde Mittel die strukturierte Darstellung in allen Projektabschnitten.

Alle bisherigen Graphiken oder graphischen Hilfsmittel erlaubten keine hinreichende verstaendliche Beschreibung der Entwurfsteile, sondern bestanden nur aus Abkuerzungen in umrandeten Feldern. Die Textbeschreibung fand immer ihre Begrenzung in sogenannten "Kaestchen". Um diese Nachteile zu vermeiden, benutzt COMPASS dafuer die Klammer-Struktogramme. In diesen Klammer-Struktogrammen werden beschriebene Strukturteile von der Aufgabenstellung bis hin zum Programmsystem mit Elementen der ingenieurmaessigen Entwurfssprache DSL verbunden und uebersichtlich dargestellt. D. h., ein Sachverhalt kann mit wenigen Worten und einfacher Symbolik schneller, uebersichtlicher und eindeutiger dargestellt werden als nur mit Worten allein. Man kann dies die technischen Zeichnungen des Software-Ingenieurs nennen. Diese Klammer-Struktogramme werden vom Rechner unterstuetzt erstellt, sind daher leicht aenderbar und koennen umgewandelt und weiterverarbeitet werden.

Die COMPASS-Werkzeuge fuegen sich gut in vorhandene rechnergestuetze Arbeitstechniken ein oder ermoeglichen neuartige, rationelle Arbeitstechniken. Dies bewirkt die Struktur der COMPASS-Werkzeuge: sie bestehen aus einzelnen unabhaengigen, jedoch aufeinander abgestimmten Programmen, und sie sind einsetzbar fuer alle Zielrechner und alle gaengigen Programmiersprachen.
Die COMPASS-Werkzeuge sind betriebssystem-unabhaengig entwickelt und daher leicht portabel.

2. Das Klammer-Struktogramm

Ein Klammer-Struktogramm wird von links nach rechts entwickelt und verfeinert, wie es auch das Bild 1 zeigt. Links von der Klammer steht der Bezeichner, rechts davon die Einzelheiten, die Details, die untereinander mit Konnektoren strukturiert verbunden sind. Jede Klammerebene hat den gleichen geschilderten Aufbau.
Die Anzahl der Konnektortypen ist sehr gering und auf einer Seite im Anhang zusammengestellt. Diese geringe Anzahl der Konnektortypen ermoeglicht es, auch ungeuebten Benutzern der Klammer-Struktogramme diese in relativ kurzer Zeit zu lesen, d. h., die Klammer-Struktogramme sind auch als Kundendokumentation nicht nur tauglich, sondern auch hervorragend geeignet.

Es gibt Konnektoren fuer
- Sequenzen,
- Selektionen,
- Parallele Aktionen,
- Iterationen.

Technische Probleme sind nicht immer so klein wie dieses Beispiel, das hier auf eine Seite passt, sondern sie haben einen groesseren Umfang und besitzen sehr viel groessere Strukturen, die nicht immer den ueblichen Darstellungsformaten entsprechen. Abhaengig von der gewaehlten Darstellungsbreite werden groessere Klammer-Struktogramme automatisch geteilt (siehe Anlage). Man erhaelt ein Hauptteil mit sogenannten Teilbildern, die numeriert und mit Verweisen dargestellt sind. Auf diese Weise gelingt es, komplexe Darstellungen zu bewaeltigen.

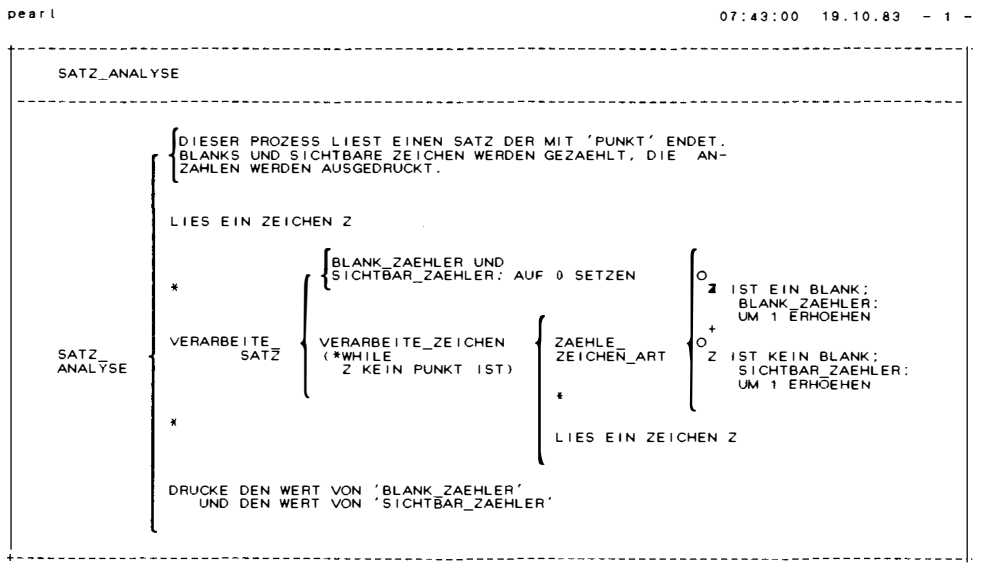


Bild 1 Klammer-Struktogramm

Nicht nur die "automatische" Teilung ist moeglich, sondern auch eine "logische" Teilung, die der Benutzer selbst steuern kann, vor allem dort, wo es logische Einheiten separat darzustellen gilt.

Das Klammer-Struktogramm wird mit dem systemeigenen Editor, den der Benutzer kennt, in alph-numerischer Form eingegeben. Das folgende Bild 2 zeigt eine sogenannte COMPASS-Quelle.

In dieser Textdatei sind die Klammern durch spezielle Zeichen am Anfang und am Ende markiert. Wie man sieht, ist diese Eingabeform recht leicht erlernbar. Man hat mit dem

Klammer-Struktogramm ein graphisch orientiertes Mittel, um Textteile strukturiert uebersichtlich darzustellen. Man gewinnt einen guten Ueberblick ueber den Verfeinerungsgrad bei der Aufgabenstellung, beim Grob- und Feinentwurf und schliesslich bei der Programmierung.

```
SATZ_
ANALYSE:
<
  DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET.
  BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT, DIE  AN-
  ZAHLEN WERDEN AUSGEDRUCKT.:
  LIES EIN ZEICHEN Z;
* VERARBEITE_
        SATZ:
  <
    BLANK_ZAEHLER UND
    SICHTBAR_ZAEHLER: AUF 0 SETZEN:
    VERARBEITE_ZEICHEN
    (*WHILE
      Z KEIN PUNKT IST);
    <
      ZAEHLE_
      ZEICHEN_ART:
      <
       ?Z IST EIN BLANK;
        BLANK_ZAEHLER:
        UM 1 ERHOEHEN;
      +?Z IST KEIN BLANK;
        SICHTBAR_ZAEHLER:
        UM 1 ERHOEHEN;
      >
    * LIES EIN ZEICHEN Z;
    >
  >
* DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
    UND DEN WERT VON 'SICHTBAR_ZAEHLER':
>
```

Bild 2 COMPASS-Quelle

3. Uebersicht ueber die COMPASS-Operatoren

3.1 Der COMPASS-Basis-Operator CSS

Der COMPASS-Basis-Operator CSS bewaeltigt folgende Aufgaben:

- Erzeugung von Klammer-Struktogrammen

In dem bereits dargestellten Bild 2 kann man wesentliche Merkmale der mit einem Editor eingegebenen COMPASS-Quelle erkennen, in der Klammeranfang und -ende sowie die einzelnen Textbloecke mit ASCII-Sonderzeichen markiert sind. Daraus entstehen die Klammer-Struktogramme. In diesen werden formatfreie Texte strukturiert dargestellt fuer die Aufgabenstellung, die Spezifikation und den Feinentwurf. Der Feinentwurf enthaelt alle wesentlichen Angaben fuer die Codierung. Die Maechtigkeit der gewaehlten Programmiersprache bestimmt den erforderlichen Verfeinerungsgrad dieser Entwurfsstufe.

```
SATZ_
ANALYSE
begin

   let DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET
       BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT, DIE  AN-
       ZAHLEN WERDEN AUSGEDRUCKT

   LIES EIN ZEICHEN Z

   VERARBEITE_
         SATZ
   begin

      let BLANK_ZAEHLER UND
          SICHTBAR_ZAEHLER: AUF 0 SETZEN

      do (*WHILE
           Z KEIN PUNKT IST)
      loop
         VERARBEITE_ZEICHEN
         begin

            ZAEHLE_
            ZEICHEN_ART
            if Z IST EIN BLANK
            then
               BLANK_ZAEHLER;
               UM 1 ERHOEHEN

            elsif Z IST KEIN BLANK
            then
               SICHTBAR_ZAEHLER;
               UM 1 ERHOEHEN

            endif

            LIES EIN ZEICHEN Z

         end

      endloop
   end

   DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
      UND DEN WERT VON 'SICHTBAR_ZAEHLER'

end
```

Bild 3 "leere" Uniquelle

```
! SATZ_
! ANALYSE
! begin
   MODULE(ANALYSE);
   SYSTEM;
      DIASGT : SGT001;
   PROBLEM;
      SPC DIASGT DATION INOUT ALPHIC DIM(*,) TFU MAX CONTROL (ALL) GLOBAL;
   ANALYSE:TASK GLOBAL;
   DCL (BLANKZAEHL,SICHTZAEHL) FIXED(15);
   DCL (Z,BLANK,PUNKT) CHAR INIT (' ','.');
!*---
!---- let DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET
!------- BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT, DIE  AN-
!-------- ZAHLEN WERDEN AUSGEDRUCKT.
!*---
!---- LIES EIN ZEICHEN Z
      GET Z FROM DIASGT BY SKIP, LIST;
!*---
!---- VERARBEITE_
!---------- SATZ
!---- begin
      BEGIN
!*---
!------- let BLANK_ZAEHLER UND
!---------- SICHTBAR_ZAEHLER: AUF 0 SETZEN
         BLANKZAEHL := 0;
         SICHTZAEHL := 0;
!*---
!------- do (*WHILE
!---------- Z KEIN PUNKT IST)
         WHILE Z NE PUNKT REPEAT;
!*------   loop
!---------- VERARBEITE_ZEICHEN
!---------- begin
            BEGIN
!*---
!------------ ZAEHLE_
!------------ ZEICHEN_ART
!------------ if Z IST EIN BLANK
               IF Z EQ BLANK THEN
!*---------- then
!---------- BLANK_ZAEHLER;
!---------- UM 1 ERHOEHEN
               BLANKZAEHL := BLANKZAEHL+1;
!*---
!------------ elsif Z IST KEIN BLANK
               ELSE
!*---------- then
!---------- SICHTBAR_ZAEHLER;
!---------- UM 1 ERHOEHEN
               SICHTZAEHL := SICHTZAEHL+1;
!*---
!------------ endif
               FIN;
!*---
!------------ LIES EIN ZEICHEN Z
               GET Z FROM DIASGT BY SKIP, LIST;
!*---
!---------- end
            END;
!*---
!------- endloop
!------- end
         END;/* REPEAT */
         END;
!*---
!---- DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
!------- UND DEN WERT VON 'SICHTBAR_ZAEHLER'
      PUT  BLANKZAEHL, SICHTZAEHL TO DIASGT BY SKIP,2 LIST;
!*---
! end
   END; /* TASK */
   MODEND;
!*---
```

Bild 4 "gefuellte" Uniquelle

- Pseudocode

Im CSS-Operator ist der Pseudocode-Generator integriert. Er erzeugt aus der fuer das Klammer-Struktogramm aufbereiteten Datei Pseudocode und legt diesen in der "Uniquelle" ab. Bild 3 zeigt die "leere" Uniquelle. Dies ist die sogenannte Linearisierung des Klammer-Struktogramms. In dieser Uniquelle wird zu dem bereits abgelegten Pseudocode mit dem Editor der Code der gewaehlten Programmiersprache eingefuegt.
In der Uniquelle findet somit die Verzahnung von Entwurf und Code statt. In Bild 4 ist ein Auszug der mit Programmmcode "gefuellten" Uniquelle dargestellt.

Ein besonderer Vorteil ist die unmittelbare Naehe des Pseudocodes - die Programmbeschreibung - und des Programmcodes. Bei Programmkorrekturen, die sich geringfuegig auf die Beschreibung auswirken, kann man gleich im Pseudocode korrigieren. (Vollstaendige Uniquelle s. Anhang).

- Compile-Fassung

Der CSS-Operator erzeugt aus der Uniquelle die entsprechende Compile-Fassung zum Uebersetzen auf dem Entwicklungsrechner oder Zielrechner. In der Compile-Fassung wird der Pseudocode als Kommentar sprachrichtig ausgegeben(s. Anhang). Zusaetzlich gibt es eine verkuerzte Compile-Fassung, naemlich den Codeextrakt, der nur wesentliche Teile des Pseudocodes zur Orientierung enthaelt und in Bild 5 dargestellt ist.

```
/*- ANALYSE */
   MODULE(ANALYSE);
   SYSTEM;
      DIASGT : SGT001;
   PROBLEM;
      SPC DIASGT DATION INOUT ALPHIC DIM(*) TFU MAX CONTROL (ALL) GLOBAL;
   ANALYSE:TASK GLOBAL;
   DCL (BLANKZAEHL,SICHTZAEHL) FIXED(15);
   DCL (Z,BLANK,PUNKT) CHAR INIT (' ','.','.');
      GET Z FROM DIASGT BY SKIP, LIST;
/*---- VERARBEITE_ */
/*------------ SATZ */
      BEGIN
         BLANKZAEHL := 0;
         SICHTZAEHL := 0;
         WHILE Z NE PUNKT REPEAT;
/*------------ VERARBEITE_ZEICHEN */
            BEGIN
/*--------------- ZAEHLE_ */
/*--------------- ZEICHEN_ART */
               IF Z EQ BLANK THEN
                  BLANKZAEHL := BLANKZAEHL+1;
               ELSE
                  SICHTZAEHL := SICHTZAEHL+1;
               FIN;
               GET Z FROM DIASGT BY SKIP, LIST;
            END;
         END;/* REPEAT */
         END;
      PUT BLANKZAEHL SICHTZAEHL TO DIASGT BY SKIP,2 LIST;
   END;/* TASK */
   MODEND;
```

Bild 5 Codeextrakt

Compile-Fassungen sind moeglich fuer alle gaengigen Programmiersprachen: wie FORTRAN, PASCAL, PL1, PL1-Varianten, PEARL, BCPL, ADA, COBOL u. a. m.

- Aenderungsmoeglichkeiten

Waehrend der Codierung, der Inbetriebnahme, des Testes und der Wartung werden Korrekturen erforderlich. Codekorrekturen und einfache Entwurfskorrekturen fuehrt man zweckmaessigerweise in der Uniquelle aus. Fuer groessere Entwurfskorrekturen bietet sich die COMPASS-Quelle an. Dazu benutzt man die Umkehrfunktion des COMPASS-Basis-Operators CSS. Mit dieser Umkehrfunktion kann man die Uniquelle, die Pseudocode und eingetragenen Programmcode enthaelt, in die COMPASS-Quelle umwandeln, ohne dass der Programmcode verloren geht. Dort korrigiert man den Entwurf, erzeugt bei Bedarf die entsprechenden neuen Klammer-Struktogramme und eine neue Uniquelle. In der neuen Uniquelle ist dann an den Stellen, an denen der Entwurf ergaenzt wurde, neuer Pseudocode entstanden, der wiederum mit Programmcode gefuellt werden kann. Auf diese Weise kann man iterativ ein Programmodul oder Programmsystem ergaenzen, verfeinern, testen oder warten.

3.2 COMPASS-Basis-Operator-Erweiterung: Rahmencode-Generator

Der Rahmencode-Generator erzeugt aus dem in der Uniquelle vorhandenen Pseudocode in der gewaehlten Programmiersprache die Steuerungskonstrukte und Zuweisungen, naemlich den Rahmencode. Dieser Rahmencode wird wie bisher neben dem Pseudocode an der richtigen Stelle in der Uniquelle abgelegt, so dass er dort vom Programmierer ergaenzt werden kann. Z. Zt. ist der Rahmencode-Generator fuer folgende Programmiersprachen vorgesehen: PASCAL, PEARL, BCPL und strukturiertes FORTRAN: RATFOR.

3.3 Die COMPASS-Dokumentations-Operatoren

- BOOK-Operator

In dem Book-Operator ist ein sehr leistungsfaehiger Text-Formatierer integriert, der die wesentlichen Aufgaben der Textbearbeitung uebernimmt. Der Book-Operator gestattet weiterhin neben der Textformatierung auch die formatierte Ausgabe von Klammer-Struktogrammen. Ebenso werden Texte und Klammer-Struktogramme gemeinsam formatiert ausgegeben. Dadurch ist es moeglich, komplette Dokumente auf einem Ausgabemedium zu erstellen.
Die Ausgaben sind auf Terminals oder auf Druckern mit oder ohne Graphikeinrichtungen moeglich. Die einzelnen

Teile eines Dokumentes koennen in verschiedenen Dateien abgelegt und von verschiedenen Mitarbeitern bearbeitet werden. Mit dem Book-Kommando koennen alle Teile dann zu einem geeigneten Zeitpunkt zu einem Dokument vereinigt werden.

- Glossar-Operator

Der Glossar-Operator sucht Stichworte aus Klammer-Struktogrammen und Uniquellen heraus, auf Wunsch getrennt fuer Code und Pseudo-Code, und bietet sie sortiert in einer besonderen Datei zur Kommentierung an. Die jeweils neuesten Begriffe sind besonders gekennzeichnet. Der Benutzer kann entscheiden, ob dieser Begriff kommentarwuerdig ist. In diesem Falle schreibt er den zugehoerigen Text unformatiert in diese Datei. Auch nicht kommentierte Begriffe koennen auf Wunsch im formatierten Text beruecksichtigt werden. Diese Glossar-Datei wird ueber den Formatierer ausgegeben, so dass die Suchbegriffe sich deutlich hervorheben und der Definitionsteil als Textblock eingerueckt vorhanden ist.

3.4 Der COMPASS-Analyse-Operator CFX

- Aufrufbaeume

Der COMPASS-Operator CFX analysiert die Aufruf-Hierarchie und stellt sie graphisch oder nach Wahl auch tabellarisch dar. Die Aufruf-Hierarchie zeigt die Abhaengigkeit der einzelnen Strukturteile untereinander.

- "cross-reference"

Der COMPASS-Operator CFX erstellt aus den angewaehlten Klammer-Struktogrammen "cross-reference"-Listen und auf Wunsch auch invertierte "cross-reference"-Listen.

- Verwendungsnachweis

Bei erweiterten Notationen in der COMPASS-Quelle kann deren Verwendung analysiert und im Zusammenhang mit den "cross-reference"-Listen dargestellt werden. Z. B. wird erkannt, ob auf Variable oder Externals lesend oder schreibend zugegriffen wird u. a. m.

- Bearbeitung der Top- und Bottom-Komponenten

Benutzt man die Top- und Bottom-Komponenten der Klammer-Struktogramme in einem Projekt systematisch, so lassen sich ueber den Analyse-Operator die Top- und/oder Bottom-Komponenten selektiv zusammenfassen. Benutzt man z. B. die Top-Komponente zur Kurzbeschreibung, so erhaelt man eine Zusammenfassung aller Kurzbeschreibungen. Verwendet man die Bottom-Komponente z. B. zur Statusbeschreibung, so erstellt man einen Statusreport des Projektes bzw. des Projektteils.

- "String"-Suchen

Der Analyse-Operator stellt eine "String"-Suchfunktion zur Verfuegung, mit der man vorgebbare "Zeichen-Strings" heraussuchen kann. Sucht man solche "Strings" in Klammer-Struktogrammen der Aufgabenstellung und in Klammer-Struktogrammen der Feinspezifikation, so kann man feststellen, ob von der Aufgabenstellung her Uebereinstimmungen bei der Implementation vorhanden sind.

4. Verwendung der COMPASS-Werkzeuge

4.1 COMPASS-Werkzeuge in den Projektphasen

In den Projektphasen koennen die COMPASS-Operatoren vielseitig eingesetzt werden.

Fuer die Aufgabenstellung (Pflichtenheft) und den Systementwurf erzeugen die COMPASS-Operatoren BOOK und CSS formatierte Texte und Klammer-Struktogramme auch in einem Dokument.

Fuer den Feinentwurf und die Codierung erstellen die COMPASS-Operatoren BOOK und CSS Texte, Klammer-Struktogramme, Glossare, Pseudocode und compilierfaehige Uniquellen. Mit dem Analyse-Operator CFX erhaelt man Aufrufbaeume, Verwendugslisten und "cross-reference"-Listen.

Fuer Test und Wartung ist die Umkehrfunktion, die die Rueckwandlung der Uniquelle in die COMPASS-Quelle bewirkt, ein wesentliches Hilfsmittel. Die Klammer-Struktogramme des Feinentwurfes, in dem Input- und Output-Daten spezifiziert sind, koennen als solide Grundlage zur Erstellung der Testplaene und Testdaten herangezogen werden. Waehrend der Wartung ist die wirkungsvolle Unterstuetzung bei Aenderungen von grossem Vorteil.

4.2 COMPASS fuer Standard-Software-Produkte

Bei Standard-Produkten gibt es unveraenderliche und veraenderliche Teile, die aus Optionen bestehen. Mit den COMPASS-Werkzeugen kann man leicht eine aktuelle Kundenvariante eines Standard-Produktes erhalten, indem man sich vorteilhaft der COMPASS-Operatoren bedient, die das Einfuegen von Moduln und die Aktualisierung der Dokumentation bewirken und eine compilierfaehige Fassung des Programmcodes herstellen.

5. Schlussbemerkung

Wie bisher gezeigt wurde, sind die COMPASS-Operatoren in allen Bearbeitungsebenen einsetzbar und in die meisten Arbeitstechniken leicht integrierbar. Die COMPASS-Werkzeuge sind von ihrer Konzeption her Werkzeuge im Sinne von "UNIX-Tools", d. h. sie benoetigen keinen eigenen COMPASS-Prozessor. Sie bearbeiten Dateien und legen die Ergebnisse wieder in Dateien ab. Daher ist es auch moeglich, andere Operatoren zur Verarbeitung dieser Dateien hinzuzuziehen.

Die COMPASS-Operatoren sind in FORTRAN geschrieben und mit relativ geringem Aufwand auf andere Rechner uebertragbar.

Die COMPASS-Operatoren sind bisher lauffaehig auf

UNIX-Systemen,
DEC VAX-11 (VMS),
DEC PDP-11 (RSX-11/M),
ATM 78 (MAX IV),
Data General Eclipse (AOS).

Die COMPASS-Werkzeuge wurden nicht nach der Massgabe entwickelt: Erschweren etwas Falsches zu tun, sondern nach der Devise: Erleichtern das Richtige zu tun.

Literatur

[1] Warnier, J.D. Logical Construction of Programs
Van Nostrand, New York 1977

[2] Orr, K. Structured System Development
Yourdon Press, New York 1977

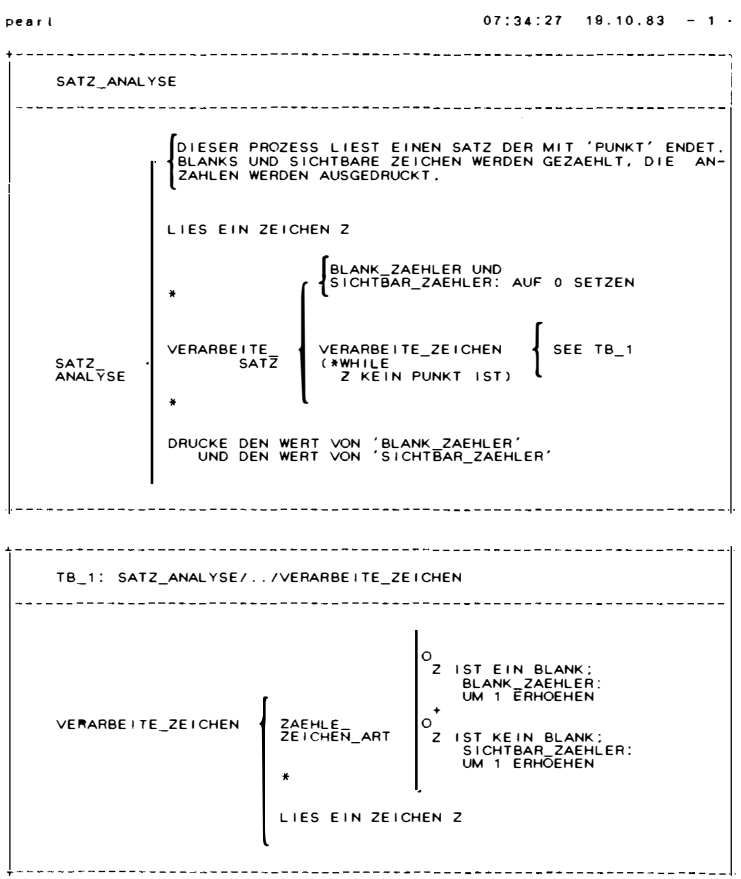
[3] Jackson, M. Grundsaetze des Programmentwurfs
stmv, Darmstadt 1979

[4] Ichbiah, J.D. u.a. Preliminary ADA Reference Manual
Sigplän Notices, Volume 14, No. 6,
June 1979

[5] Ehling, H.J. Datenstrukturierter Entwurf fuer Echtzeit-Systeme
PVD-Berichte KfK-PDV (Hommel Ed.)

[6] Ehling, H.J. Evolutionaerer Systementwurf
Informatik-Fachberichte 21,
pp 69-93
Springer Berlin 1979

ANHANG I Geteiltes Klammer-Struktogramm



ANHANG II COMPASS-Quelle mit Programmcode

```
SATZ_
ANALYSE:
<
  $FMODULE(ANALYSE);
    SYSTEM;
       DIASGT : SGT001;
    PROBLEM;
       SPC DIASGT DATION INOUT ALPHIC DIM(,) TFU MAX CONTROL (ALL) GLOBAL;
    ANALYSE:TASK GLOBAL;
    DCL (BLANKZAEHL,SICHTZAEHL) FIXED(15);
    DCL (Z,BLANK,PUNKT) CHAR INIT (' ',' ','.');$$
  DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET.
  BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT. DIE  AN-
  ZAHLEN WERDEN AUSGEDRUCKT.
  $F@@@$$;
```

```
   LIES EIN ZEICHEN Z
   $FGET Z FROM DIASGT BY SKIP, LIST;$$:
 * VERARBEITE_
         SATZ;
   <
     $FBEGIN$$
     BLANK_ZAEHLER UND
     SICHTBAR_ZAEHLER: AUF 0 SETZEN
     $FBLANKZAEHL := 0;
       SICHTZAEHL := 0;$$:
     VERARBEITE_ZEICHEN
     (*WHILE
       Z KEIN PUNKT IST)
     $FWHILE Z NE PUNKT REPEAT;$$;
     <
       $FBEGIN$$
       ZAEHLE_
       ZEICHEN_ART:
       <
        ?Z IST EIN BLANK
         $FIF Z EQ BLANK THEN$$;
         BLANK_ZAEHLER:
         UM 1 ERHOEHEN
         $FBLANKZAEHL := BLANKZAEHL+1;$$;
       +?Z IST KEIN BLANK
         $FELSE$$;
         SICHTBAR_ZAEHLER:
         UM 1 ERHOEHEN
         $FSICHTZAEHL := SICHTZAEHL+1;$$;
       >
       $FFIN;$$
     * LIES EIN ZEICHEN Z
       $FGET Z FROM DIASGT BY SKIP, LIST;$$;
     >
     $FEND;$$
   >
   $FEND;/* REPEAT */
     END;$$
 * DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
     UND DEN WERT VON 'SICHTBAR_ZAEHLER'
   $FPUT  BLANKZAEHL, SICHTZAEHL TO DIASGT BY SKIP,2 LIST;$$;
>
$FEND;/* TASK */
  MODEND;
  $$
```

ANHANG III ungekuerzte "gefuellte" Uniquelle

```
|- SATZ_
|- ANALYSE"
|- begin
   MODULE(ANALYSE);
   SYSTEM;
      DIASGT : SGT001;
   PROBLEM;
      SPC DIASGT DATION INOUT ALPHIC DIM(,) TFU MAX CONTROL (ALL) GLOBAL;
   ANALYSE:TASK GLOBAL;
   DCL (BLANKZAEHL,SICHTZAEHL) FIXED(15);
   DCL (Z,BLANK,PUNKT) CHAR INIT (' ',' ','.');
|*---
|---- let DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET.
|-------- BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT. DIE  AN-
|-------- ZAHLEN WERDEN AUSGEDRUCKT.
|*---
|---- LIES EIN ZEICHEN Z
      GET Z FROM DIASGT BY SKIP, LIST;
|*---
|---- VERARBEITE_"
|---------- SATZ"
|---- begin
      BEGIN
|*---
|------- let BLANK_ZAEHLER UND
|---------- SICHTBAR_ZAEHLER: AUF 0 SETZEN
         BLANKZAEHL := 0;
         SICHTZAEHL := 0;
|*---
|------- do (*WHILE
|------------ Z KEIN PUNKT IST)
         WHILE Z NE PUNKT REPEAT;
|*------ loop
|---------- VERARBEITE_ZEICHEN"
|---------- begin
            BEGIN
|*---
|------------- ZAEHLE_"
|------------- ZEICHEN_ART"
|------------- if Z IST EIN BLANK
                IF Z EQ BLANK THEN
|*------------ then
|---------------- BLANK_ZAEHLER:
|---------------- UM 1 ERHOEHEN
                   BLANKZAEHL := BLANKZAEHL+1;
|*---
|------------- elsif Z IST KEIN BLANK
                ELSE
|*------------ then
|---------------- SICHTBAR_ZAEHLER:
|---------------- UM 1 ERHOEHEN
                   SICHTZAEHL := SICHTZAEHL+1;
|*---
|------------- endif
                FIN;
|*---
|------------- LIES EIN ZEICHEN Z
                GET Z FROM DIASGT BY SKIP, LIST;
|*---
|---------- end
            END;
|*---
|------- endloop
|------- end
         END;/* REPEAT */
         END;
|*---
|---- DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
|------- UND DEN WERT VON 'SICHTBAR_ZAEHLER'
      PUT  BLANKZAEHL, SICHTZAEHL TO DIASGT BY SKIP,2 LIST;
|*---
|- end
   END;/* TASK */
   MODEND;

|*---
```

ANHANG IV Compile-Fasung der Uniquelle

```
/*|- SATZ_
/*|- ANALYSE"
/*|- begin|*/
     MODULE(ANALYSE);
     SYSTEM;
        DIASGT : SGT001;
     PROBLEM;
        SPC DIASGT DATION INOUT ALPHIC DIM(,) TFU MAX CONTROL (ALL) GLOBAL;
     ANALYSE:TASK GLOBAL;
     DCL (BLANKZAEHL,SICHTZAEHL) FIXED(15);
     DCL (Z,BLANK,PUNKT) CHAR INIT (' ',' ','.');
/*|*---
/*|---- let DIESER PROZESS LIEST EINEN SATZ DER MIT 'PUNKT' ENDET.
/*|-------- BLANKS UND SICHTBARE ZEICHEN WERDEN GEZAEHLT. DIE  AN-
/*|-------- ZAHLEN WERDEN AUSGEDRUCKT.
/*|*---
```

```
/*|---- LIES EIN ZEICHEN Z|*/
       GET Z FROM DIASGT BY SKIP, LIST;
/*|*---
/*|---- VERARBEITE_^
/*|----------- SATZ^
/*|---- begin|*/
       BEGIN
/*|*---
/*|------- let BLANK_ZAEHLER UND
/*|----------- SICHTBAR_ZAEHLER: AUF 0 SETZEN|*/
          BLANKZAEHL := 0;
          SICHTZAEHL := 0;
/*|*---
/*|------- do (*WHILE
/*|------------ Z KEIN PUNKT IST)|*/
          WHILE Z NE PUNKT REPEAT;
/*|*------ loop
/*|---------- VERARBEITE_ZEICHEN^
/*|---------- begin|*/
              BEGIN
/*|*---
/*|------------- ZAEHLE_^
/*|------------- ZEICHEN_ART^
/*|------------- if Z IST EIN BLANK|*/
                 IF Z EQ BLANK THEN
/*|*------------ then
/*|---------------- BLANK_ZAEHLER:
/*|---------------- UM 1 ERHOEHEN|*/
                    BLANKZAEHL := BLANKZAEHL+1;
/*|*---
/*|------------- elsif Z IST KEIN BLANK|*/
                 ELSE
/*|*------------ then
/*|---------------- SICHTBAR_ZAEHLER:
/*|---------------- UM 1 ERHOEHEN|*/
                    SICHTZAEHL := SICHTZAEHL+1;
/*|*---
/*|------------- endif|*/
                 FIN;
/*|*---
/*|------------- LIES EIN ZEICHEN Z|*/
                 GET Z FROM DIASGT BY SKIP, LIST;
/*|*---
/*|---------- end|*/
              END;
/*|*---
/*|------- endloop
/*|------- end|*/
          END;/* REPEAT */
          END;
/*|*---
/*|---- DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
/*|------- UND DEN WERT VON 'SICHTBAR_ZAEHLER'|*/
       PUT  BLANKZAEHL, SICHTZAEHL TO DIASGT BY SKIP,2 LIST;
/*|*---
/*|- end|*/
    END;/* TASK */
    MODEND;

/*|*---|*/
```

ANHANG V

Glossar zu: SATZ_ANALYSE 16.10.83 - 1 -

GLOSSAR zu SATZ_ANALYSE

BLANK	Blanks sind Leerzeichen. Sie werden gesondert gezaehlt.
BLANK_ZAEHLER	zaehlt die Leerzeichen. -> BLANK.
PUNKT	beendet einen Satz. Wird vom Programm erkannt.
SATZ	wird analysiert.
SICHTBAR_ZAEHLER	zaehlt die sichtbaren Zeichen, jedoch keine Leerzeichen.
VERARBEITE_SATZ	Hier geschieht die eigentliche Analyse in diesem Programm.
VERARBEITE_ZEICHEN	ist jeweils die Analyse eines einzelnen Zeichens.
Z	Z ist das jeweils verarbeitete Zeichen.

ANHANG VI Suche nach Zeichen - 1 -

SATZ_ANALYSE

SUCHE NACH ZEICHEN

Es wird nach den Strings "BLANK_ZAEHLER" und "SICHTBAR_ZAEHLER" gesucht.

```
/SATZ_ANALYSE
  |
  |  BLANK_ZAEHLER
  |  SICHTBAR_ZAEHLER
```

SUCHE, QUERBEZUGSLISTE

Es wird die Querbezugsliste fuer die ausgewaehlten Strings "BLANK_ZAEHLER" und "SICHTBAR_ZAEHLER" erstellt.

```
BLANK_ZAEHLER
  |
  |  /SATZ_ANALYSE
  |  /SATZ_ANALYSE/VERARBEITE_SATZ
  |  /SATZ_ANALYSE/VERARBEITE_SATZ/VERARBEITE_ZEICHEN/ZAEHLE_
 -|-  ZEICHEN_ART

SICHTBAR_ZAEHLER
  |
  |  /SATZ_ANALYSE
  |  /SATZ_ANALYSE/VERARBEITE_SATZ
  |  /SATZ_ANALYSE/VERARBEITE_SATZ/VERARBEITE_ZEICHEN/ZAEHLE_
 -|-  ZEICHEN_ART
```

BLATTAUSGABE

Es werden die Textbloecke ausgegeben, in denen einer der String "BLANK_ZAEHLER" oder "SICHTBAR_ZAEHLER" vorkommt.

```
/SATZ_ANALYSE/VERARBEITE_SATZ
  |
  |  BLANK_ZAEHLER UND
  |  SICHTBAR_ZAEHLER: AUF 0 SETZEN

/SATZ_ANALYSE/VERARBEITE_SATZ/VERARBEITE_ZEICHEN/ZAEHLE_ZEICHEN_
ART
  |
  |  BLANK_ZAEHLER:
  |  UM 1 ERHOEHEN
  |  SICHTBAR_ZAEHLER:
  |  UM 1 ERHOEHEN

/SATZ_ANALYSE
  |
  |  DRUCKE DEN WERT VON 'BLANK_ZAEHLER'
  |     UND DEN WERT VON 'SICHTBAR_ZAEHLER'
```