O N L I N E

D A T A B A S E

START TO FINISH

Robert B. Garvey

Robert L. Womack IV

Witan Inc,
Kansas City, Missouri

Presentation Outline

1   Introduction, what will be covered.

A   The Foundations:

   1   Goals; A System Language and Methodology

   2   System Principles

      A   Elements
        1   Components
        2   Relationships

      B   Use in System Phases
        1   Analysis 2   File Design 3   General Design

   3   Information System Architecture

      A   General System Architecture
        1   Detailing 2   Development 3 Implementation

      B   Use of Image and View

   4   Interactivity and Control

      A   Menu Programs

      B   Control Tables

      C   Data Area Control

      D   Quiet Callability

B   Dynamically Callable Programs:

   1   SL's USL's

1. Introduction

Foundation

Bob Garvey will first lay the a foundation for the understanding of an approach used to understand, design, and implement interactive systems.

A system language, Goals, will be introduced to render systems and components.

A general set of principles will be presented incorporating the components and structures inherent in a structured system. The use of these components in the system life cycle and as a documentation system will evolve.

A general system architecture will be presented and an approach to interactivity will be discussed.

Callables

Bob Womack will present the detailed use of callable programs in the 3000 environment.

G O A L S

A System Language

Goals was designed to meet the following criteria:

* Provide good documentation
* Ease maintainence
* Expedite development
* Provide users early understanding of System functions and restraints
* Improve project management and reporting
* Reduce resources required for documentation
* Optimize System performance

Many of the above criteria can be achieved through reasonable structuring of the system . But many of the structuring techniques that are now popular are simply more trouble than they are worth. Yourdon, Jackson and certainly IBM's HIPO involve more work in their maintainence than rewards merit. Warnier comes closest to being worthwhile but cannot be reasonably maintained in machine sensible form.

Goals will be described as a methodology only because it seems to accomplish all the criteria of the popular "Methodologies", and much more. We do not feel that any of the methodologies should be considered ends in themselves and more sacred than the system at hand. Once the principles are learned and applied the implications should be obvious and the apparent need for a methodology forgotten.

Documentation

1 General Statement
2 Goals, Structural Notation

1 General Statement

The purpose of documentation is to assist in the maintainence and operation of a system. To those ends software documentation must be flexible, easily modifiable, current and easy to read. Witan has developed a system of documentation called Goals which uses simple text files associated through control numbers to meet the criteria listed above. The following sections (2 and 3) describe the general features of the structural notation used in Goals and the General system structure used in system projects.

Goals is used throughout the life of a project. It is used to:
1 To state requirements
2 Render flow and components in the analysis phase
3 To develop, test and render a general design
4 As a pseudo code or structured english for detail design
5 As a high level programming language
6 As a project network descriptor.

2 Goals, Structural Notation

Formal structuring permit three primitive operations:
Sequence, Repetition and Alternation. Structural Notation was developed to meet the criteria of formal systems in a generalized way and was guided by the assumption that systems must be rendered in a machine sensible form. Goals relies upon text sequences and key words as it's basis. Structural Notation is the basis of the syntax of Goals.

Following are the representations of the primitive structures using flowcharts and Goals. The word process is used to represent a step, a process or an item depending on the use of the notation at the time.

Goals Primitive Structures

S E Q U E N C E
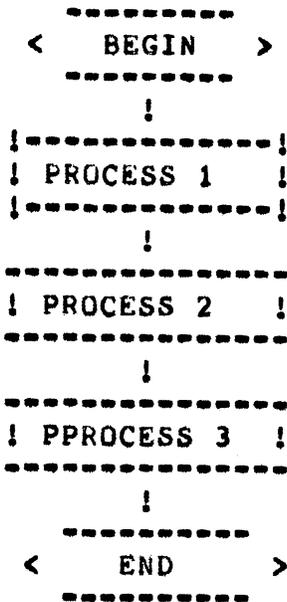
```
FLOW                 -----------
                   <    BEGIN    >
                     -----------
                          !
               !----------------!
               !  PROCESS 1     !
               !----------------!
                          !
               -----------------
               !  PROCESS 2     !
               -----------------
                          !
               -----------------
               !  PPROCESS 3    !
               -----------------
                          !
                     -----------
                   <    END      >
                     -----------


GOALS          1    PROCESS 1
               2    PROCESS 2
               3    PROCESS 3
```

A L T E R N A T I O N

FLOW

```
              -----------
          <     BEGIN     >
              -----------
                  !
                  *
              *       *
          *               *
      *       IF  X     * --- true-----> !-----------------!
          *               *               !   PROCESS 1   !---!
              *       *                   !-----------------!   !
                  *                                             !
                  !                                             !
                false                                           !
                  !                                             !
                  *                                             !
              *       *                                         !
          *               *                                     !
      *       IF  Y     * ---true-----> !-----------------!     !
          *               *               !   PROCESS 2   !---!  !
              *       *                   !-----------------!   ! !
                  *                                             ! !
                  !                                             ! !
                false                                           ! !
                  !                                             ! !
                  *                                             ! !
              *       *                                         ! !
          *               *                                     ! !
      *       IF  Z     *---true-----> !-----------------!       ! !
          *               *               !   PROCESS 3   !---!   ! !
              *       *                   !-----------------!   !  ! !
                  *                                             !  ! !
                  !                                             !  ! !
                false                                           !  ! !
                  !                                             !  ! !
                  !<--------------------------------------------<- !
                  !
              ---------
          <     end     >
              ---------
```
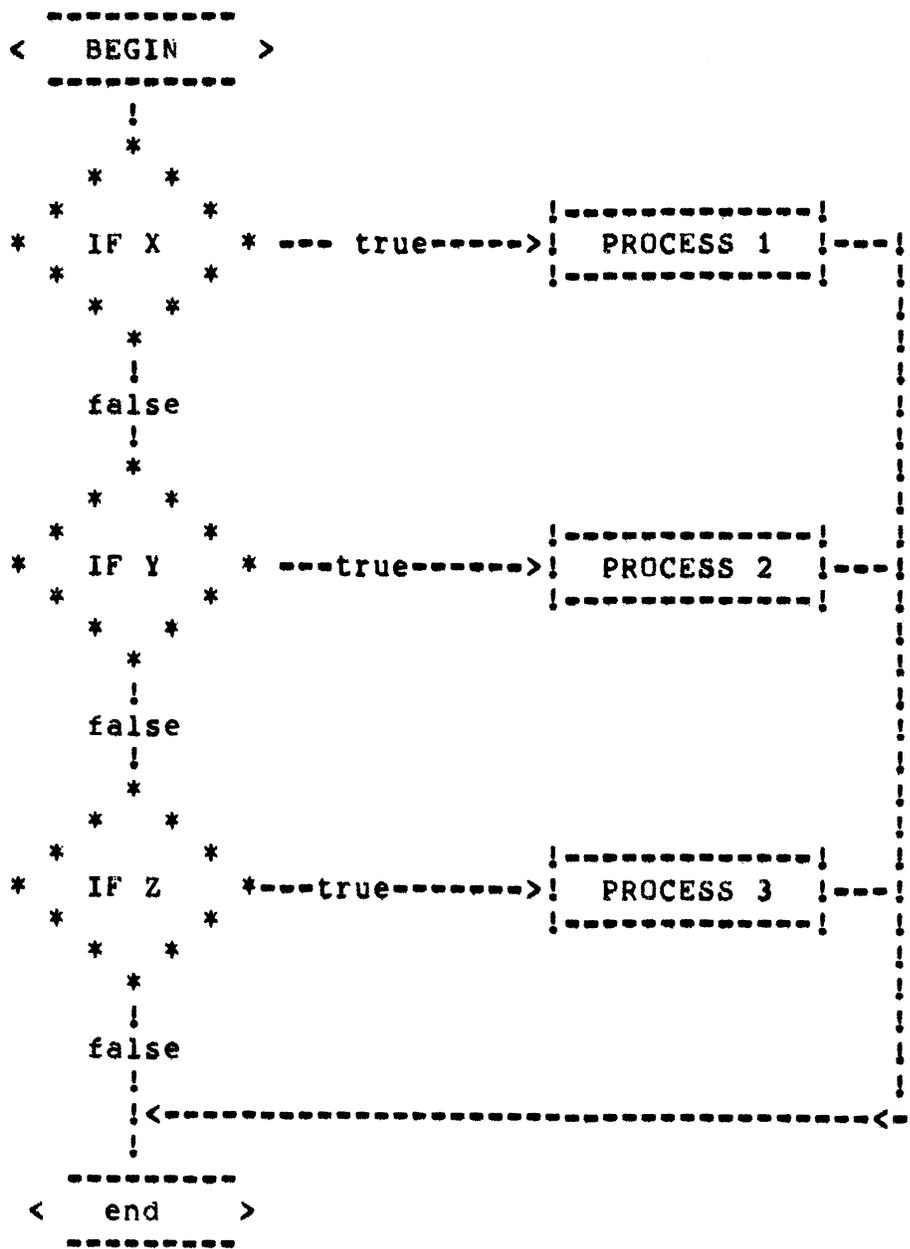
GOALS   IF X IS TRUE
                 PROCESS 1
             IF Y IS TRUE
                 PROCESS 2
             IF Z IS TRUE
                 PROCESS 3

R E P E T I T I O N

FLOW

```
                      ----------
               <      BEGIN      >
                      ----------
                          !
                          *
                      *       *
                     *           *
    <-false---<*    IF Y    *----true----
        !          *           *         !
        !           *       *            !
        !               *                !
    ------              !              !----------------!
    < END  >            !<-----------! PROCESS 1      !
    ------                            !----------------!
```

GOALS   REPEAT UNTIL Y IS FALSE
                  PROCESS 1
                      PROCESS 1A
                      PROCESS 1B
                      PROCESS 1C
                  !

The exclamation point is used to signify control in the
REPEAT loop. If the condition is met the control passes to the
statement following the (!) on the same level. If the condition
is met the control passes to the first statement following the
condition.
    Processes 1A through 1C were added to show a simple subsequence.

DATA STRUCTURING

Goals is also used to represent data structure. As with control
structure there are three general structures which can be
represented.
   Data items listed line after line represent sequence:
   1 item-1
   2 item-2
   3 item-3

Subsequences are represented as sequences on a level below the
item of which they are are a part.

   1 item-1
   1A   item-1A
   1B   item-1B
   1C   item-1C
   2 item-2
   3 item-3

LEVELS: are represented graphically with the use of indentation. The
first character in a line is considered to begin an "A" level
subsequent levels are indented an additional three spaces each.

Successively lower levels (higher value characters and more deeply
indented) represent subordinate processes. As will be seen in the
general system structure the highest most levels are controled by
increments of time; years, quarters, months, days, etc. while lower
levels are controlled by events or conditions.

CONTROL NUMBERS: The control numbers used in Goals are developed by
alternating the use of numbers and letters to represent sucessively
lower levels within the system. The system is similiar to English
outlining except that only capital letters and numeric characters are
used. For a given statement there is nothing to indicate its position
in the hierarchy unless the entire control number is dipicted or the
starting control number on the page is given. When Goals statements are
machine stored the entire control number is either stored or is
assumed.

Repetition in data structuring can be represented by "(S)" at the end
of the item name which is repeated, this can take the form of an
expression (i.e. (0>s<15)).

```
1 item-1(S)
1A    item-1
```

Example: a file of accounts

```
          Account File
1 Account(s)
1A    Account
1B    Account number
1C    Name
1D    Address(s)
1D1       Address type (h=home,w=work)
1D2       Street number
1D3       Direction
1D4       Street name
1D5       Affix
1E    Amount due
1F    Order(s)
1F1       Order number
1F2       Item(s)
1F3           Item
```

ALTERNATION:
   Alternation is represented with the IF control word or with the
notation (1,0).

```
2A    IF segment descriptive code = 1
2A1       material
2B    IF segment descriptive code = 2
2B1       supply
```

This convention is seldom used because the REPEAT handles
most situations for the case of data structuring.

The other type of alternation is within a string of data items
where the item can either exist or not exist. Another way of rep-
resentin a non-required item.

```
1 item-1
2 item-2
3 item-3(1,0)
```

This says that items 1 and 2 must exits or are required and item 3 is
optional.

Discussion:

   The highest level of repetition within a data structure is assumed to
be the key to the file or at least the major sort sequence. If

additional keys are required they can be represented with the word KEY
(i.e. item-3 (KEY)) or an additional data structure can be presented
to represent the structure represented when the KEY is used.

Goals can be used to represent logical structures as well as the
physical implementations.   It is important that the required logical
views of data be derrived and documented before any phy- sical
structures be planned. A recommended goal in system design is to have
a one to one relationship between the physical and the logical
structures of the system. The coding complexity is reduced appreciably
as well as the maintainence activity. An additional byproduct is the
ability to use Query or other general inquiry languages in a more
straight forward fashion.

2 Principles

An Information system is distinguished from operating systems, command interperters, compilers and the like. An Information System is that set of communications, operations, files and outputs associated with a single conceptual "file".

I am not talking about a single program. Historically I am talking more about an application area.

A. Elements

A1 Components

First an analogy: All purely mecanical devices are made up of elemental components; the incline plane, the wheel and axle, the lever and the chamber. The physics of these basic components and the materials from which they are constructed define the limits of their application. You may be saying, that list does not sound correct or "what about the screw". In listing elemental components certain definitions are inherent. I define the screw as a "rolled incline plane".

For information systems I assert that the list is: Communications, files, operations and outputs. The limits for such systems are defined by the ordering of the elements using the primitive structures (sequence, alternation and repetition).

As a note; to date the list of elemental components may have been input, process and output without reguard to to structure. This is more elemental considering all computer processes but is unbounded. This makes a general system design technique very difficult. Adding hierarchy to the above does not enhance these primitives to any great extent.

A2 Relationships

With these boundries and definitions in hand, lets look at the relationships that develop.

There is generally a one to one relationship between file structure and operations structure, between communications structure and operations structure, between output structure and operations structure. In other words the operations or control structure mimics the other components of the system and each componet is related to the other in structure. The structure begin with the file structure.

Example; if you have a file of accounts and you want to report them; the report program will have to be structured exactly the same as the file or database to report all the data in the file. Most often there is a one to one relationship between files and outputs. In the report example the report structure could be expected to look exactly like the file. If the report is to look different than the file there would be in intervening operation usually a sort or selection to convert an

intermediate output to the final output.

The same is true of communications which on the data processing level are the transmissions to the uses, the screens and the messages. The structure of a communication is generally the same as the operation structure which is the same as the data structure and thus the communication structure is the same as the data structure. This substantiates the theory that systems can be completly described knowing only the data structure. True but limited. Knowing the structure of any part should in theory give you the whole.

If everything describable about a system can be described in simple structures (and thus in Goals) and the components of a system include only communications, files, operations, and outputs and Goals can be used in all system phases then we have a framework for a general system covering conception through maintainence.

Lets look at any application. Traditionally you would begin with a requirements statement and do an analysis of the existing system. Forget flowcharts, classic narratives, and other charting techniques. Think of progressively decomposing the system using simple english outlining starting with the functions. Functions fit into the operations structure discussed. You will note that as you get down a level or two you will encounter repetitive tasks dependant on conditions, add REPEAT and IF to your outlines and keep describing. Remember that users can understand outlines and repetition and alternation are not difficult to understand.

Operations will include existing machine processes, manual proceedures, paper flows, sorting processes ect. As you are going through the operations keep a list of the files that are mentioned and note the file keys (and sorts) and any advantages or requests for multiple keys.

List any outputs or reports prepared by the organization or required in the future.

Communications will be minimal at this stage but note any memos that may go from one section to another of a "file" of notes used as crossreference or duplicate of any more perminent file.

Your documentation is now shaping up; your notebook and I assume that the whole world has change to 8 1/2 by 11, should be divided into communications, file, operations and outputs.

The starting point for design is the detailing of the files in your file list. You will want to reduce the files as much as possible to a single file. By way of naming conventions the "file" should have the same name as the system at hand.

You will notice that many of the manual files are really communications in that they are "views" of the file that are required in a particular subfunction.

The design of the conceptual file must be validated against the required operations. I am going to leave this hanging for a moment to discuss a General System Structure.

3 General System Structure

A General System Structure is presented on the following page in Goals.

This structure is not applicable in all systems but is used as a pattern for system discription, design and understanding.

The key elements of design of this structure are:
1. File unity; a system with this structure has only
   one conceptual file. It may have any number of
   datasets of or physical files but they must be
   formalized into one.
2. Journalizing or logging; all changes made to data
   items can be (and normally should be) logged.
3. Last action dating; incorporated as part of logging,
   permits an offline log.

   One detailed implication of this is need to have a date
   stamp in each detail set and a master date stamp in the
   master file.

General System Architecture

```
Begin system
REPEAT until EOSystem
   REPEAT until EOYear
      REPEAT until EOQuarter
         REPEAT until EOMonth
            REPEAT until EODay
               REPEAT for each user
                  Begin online
                  Identify operator and security
                  Open system file
                  Open current files
                  REPEAT for each Communication
                     IF control transfer
                        transfer control
                     IF batch request
                        initiate request
                     IF update , add or delete
                        Begin
                        Memo to LOG
                        LOCK
                        Update ,add or delete
                        UNLOCK
                        End
                     IF inquiry
                        perform communication operation

                        •
                     !
                  !
               !
               End Online
               Begin daily batch
               Perform daily batch processing
               Run LOG analysis
               If end of week
                  Perform Monthly Processing

                  •
               ROLL FILES
            !
         !
         perform Monthly processing
         Perform Quarterly processing
      !
   Perform end of year processing
!
Close system
End
```

A GENERAL DESIGN

With this Architecture and database design complete we have the basis for the development and implementation of any application.

Step 1 is inquiry into our file; if there is only one search criteria then we calculate into to file and return the master data or a summary. Once positioned in a master we can chain through our detail sets or follow appropriate programatic paths.

The master screen (a communication) should provide inquiry, update, and addition ability.

Each detail set should have a screen providing the same update add and inquiry ability. Our screens will be one for one with the detail sets. Think of a detail set as having a buffer that will correspond to communication (VIEW) buffer. Moving date within one program is facilitated with this concept.

The list of detail sets becomes a list of programs which must be written to handle the retrieval, update, addition, deletion and editing of data for the detail set.

When this is complete you will have a functioning system; it will not function well. I have intentionally oversimplified. The office proceedures which may be in place or will evolve will dictate what combination of sets will appear on a screen but no effort was be lost in developing the barebones system according to this method. Each set (detail set) should have its own program to handle retrieval and update. When requirements demand inclusion the programs can usually be used with few changes. You can take this one step further to include a general scheme to handle multiple data sets on one screen.

The question then becomes; "How do I tie this all together?".

4 Interactivity and Control

Lets say that we have written a system composed of a series of programs that correspond to our data sets. The way in which we permit interactivity is through a control program called MENU.

4A Menus

A master data set will exist at the top of the conceptual file and the primary search path will be the file key. Other search paths will be provided through subsystems such as "Name Family" or through automatic masters. For all detail sets associated to the master there will be a program to handle that data set. Your analysis will dictate all the processes that the operator may wish to perform. As other requirements develop associating more than one data set the code can be combined and new screens developed.

The menu control program provides transfer of control. It can do this either "quietly" of "loud". Loud is the obvious implementation; the operator choses a data set from a menu screen, the control is transfered via a "call" to a dynamic subprogram the data set is accessed updated, ect. and control returns to the controlling menu. But let us give the operator the ability to "tell" the system where he wants to go next. If he does a common area flag can be set to say don't display the menu simply transfer control to some other subprogram. We call are common area for data SYSBLK and out flag(s) Q1, Q2, ect. (you are not limited to one level of menu).

A menu structure may look like this:
```
      MAIN MENU
   REPEAT UNTIL PARENT OR END OF SYSTEM
      IF LOUD
         GET MAIN MENU SCREEN
         SEND (SHOW) SCREEN
         REPEAT UNTIL EDITS PASS
           EDIT FIELD
           IF EDITS FAIL
             SEND SCREEN

         •
         SET MODE TO QUIET
      IF QUIET
         IF NEXTPROCEEDURE=A
            CALL A
         IF NEXTPROCEEDURE=B
            CALL B

            •••
         IF NEXTPROCEEDURE =N
            CALL N
         ELSE
            CALL CONTROL'NUMBER'TABLE

         •
      !
```

Through this technique those programs which are not being used are not using memory resources. The CONTROL NUMBER TABLE refers to implementations which have levels of menus. If the control reference is not handled at that menu level control is appropriatly passed to the proper level where a control program can handle it.

The quiet "CALL" technique can be used for any of the data set programs discussed by putting the quiet call structure "around" the program and requiring the passing of appropriate data into or from the communication buffer. Bob Womack will describe this technique in the "NAME FAMILY" discussion.

SL's and USL's

S L ' s

o   Modules, Entry points, Programs referenced in require
     CST entries if they are not allready referenced in a
     running program.

o   Code is sharable by all programs. The PUB.SYS SL
     is avalable to all programs. Account and group SL's
     are available to programs being run out of that Account.

o   You need exclusive access to the SL to make an entry in it.

o   When SL entries are made you do not need to prepare the
     SL. It is available after you have exited the segmenter.

U S L ' s

o   Programs compiled into a USL must be prepared before they are
     runnable.

o   Many programs may be compiled into the same USL. When
     a program is run the system will look to the USL for resolution
     of called programs, it then looks to the PUB.SYS SL unless
     a library is specified in the RUN. (RUN prog;LIB=G)

o   All USL resolved entries create XCST entries except the outer
     block.

CST's and XCST's

o   There are 192 CST entries available to user processes

o   There are 1028 XCST entries available to user processes.

COMPILE INTO A USL

```
!JOB JOBNAME,username/ serpass.accountname/accountpass;OUTCLASS=,1
!COBOL progname,$NEWPAS ,$NULL
!SEGMENTER
USL $OLDPASS
NEWSEG progname,progname'
PURGERBM SEGMENT,progna e'
USL yourusl
PURGERBM SEGMENT,progna e
AUXUSL $OLDPASS
COPY SEGMENT,progname
EXIT
!TELL user.acct; yourprog ---> yourusl
!EOJ
```

P R E P   O F   U S L

```
!JOB DyourUSL,user/userpass.account/accountpass;PRI=ES;OUTCLASS=,1
!PURGE yourrun
!CONTINUE
!BUILD yourrun;DISC=2500,1,1;CODE=PROG
!SEGMENTER   .
USL yourusl
PREPARE yourrun;MAXDATA=16000;CAP=MR,DS
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

C A L L A B L E S   I N T O   S L ' s

```
!JOB D!SL,user/userpass.account/accountpass;OUTCLASS=,1
!COBOL yourprog,$OLDPASS,$NULL
!SEGMENTER
AUXUSL $OLDPASS
SL SL
ADDSL yourprog
EXIT
!TELL user.acct;  yourrun ---> yourrun
!EOJ
```

```
                        M E N U

REPEAT until parent or  nd of system
    IF load
        get menu screen
        show screen
        REPEAT until edits pass
            edit fields
            IF edit fail
                send screen
        !  .
        set mode to quiet
    .
    IF quiet
        IF nextprocedure = "O"
            CALL "O" USING ., ., .
        IF nextprocedure = "I"
            CALL "I" USING ., ., .

                .

                .

                .
        IF nextprocedure = "n"
            CALL "n" using ., ., .
        ELSE
            CALL "CONTROLNUMBERTABLE" using nextprocedure
        .
!  .    .
```