ABSTRACT

Computer-aided software engineering (CASE) tools are gaining acceptance in the Pharmaceutical community. This paper describes a system that was developed specifically for clinical trials reporting and which incorporates some CASE technologies. Using this system one can model data and create prototypes of reports in a computer-aided design mode. The scripts that define these prototypes can then be processed by a code generator to produce modular SAS programs. These code modules can be modified by using an 'update' procedure. The benefits and limitations of this system will be discussed as the reader is stepped through the process from prototype definition to final SAS program listings.

INTRODUCTION

Each year the pharmaceutical industry spends billions of dollars formulating and testing new medications. To make a solid case for the safety and effectiveness of a drug huge databases of clinical trials case report form data must be assimilated and analyzed. Federal agencies mandate that this information be submitted in formal listings when a license application is filed. The programming effort this requires is formidable. Hence, there is a real incentive to find better ways to represent data accurately, efficiently, and in a style that is aesthetically coherent and pleasing to the application review board.

Computer-aided Software Engineering, or as it is sometimes called 'Information Engineering', provides methods and tools for developers to define, model, and construct software systems. The first section of this paper "PROTOTYPING" describes a tool which allows developers to deal with report design problems and solutions at a high level of abstraction from actual SAS code. The second section "CODE GENERATION" limns the features of a tool which can greatly speed the transformation of the prototype into a successfully executing SAS application. Throughout a facsimile of a clinical report is used as an example to demonstrate the ways these CASE tools can be used to aid production of a complex and realistic SAS program. This example is provided only as a basis for discussion and is fictitious. Any resemblance to one of the dozens of reports actually produced by this system is purely coincidental. Although written in the context of clinical research the methods discussed in this paper are extensible to any industry.

PROTOTYPING

The graphic in Figure 1 illustrates the traditional way of designing and developing a report. A user or group of users hashes out the needed fields and layout. This is often based on an existing report from another study. In the typical hand-drawn 'mockup' elements such as field widths, title widths, orientations, and indents are rarely specified in sufficient detail. Much is left to the programmer's imagination and style. When the mockup is submitted as a request the programmer then tries to produce a best 'guesstimate' of what is wanted. The process then loops back to the requestors for clarification, revision, or approval. Each cycle in this loop is prone to delays and downtime. Sometimes weeks or months can be lost waiting for an approval to clear someone's desk. In such an environment miscommunications and mistakes are amplified and deadlines rather than quality can become the focus.

Lack of a robust design paradigm has other drawbacks. Cosmetic inconsistencies among reports in a large project are commonplace. As programs pass through many revisions and perhaps many different hands they can take on a 'code-by-committee' appearance. This makes the code hard to read and increases maintenance overhead. The process is fraught with redundancies and wasted
effort. Nothing in the design phase has leveraged the actual production of SAS code.

Figure 2 CASE Report Definition Model

In contrast to this model is the approach illustrated in Figure 2. In the CASE model the requestor and the programmer are brought together within the design clarification and revision loop. Using a CASE language shorthand or script a programmer is able to render a complex report prototype in minutes and then display it on-screen with What-You-See-Is-What-You-Get realism. When alterations to a design are proposed there is no need to go back to the 'drawing board'. In this system the drawing board is the keyboard, and proposed changes can be viewed immediately. When the requestors work with the programmers to sculpt the layout to their liking they both share a common understanding of the final product. The programmer now has accurate specifications rendered in a useful fashion and in precise detail.

Prototyping has additional benefits. One has to do with the practice in clinical research of conducting many similar studies in tandem or in sequence to one another. A library of stored design scripts or prototypes, therefore, can serve as templates when it comes time to work-up report designs for the next study in the series. This can be an implicit means for enforcing design standards. Besides helping to reach a design consensus quickly, the CASE approach emphasizes cooperation and teamwork. There are likely creative and qualitative synergias gained in the process. Most significantly in this system the script that defines the prototype can be used to generate the SAS code which produces the actual report. This means that whatever effort is expended in the design phase leverages the construction phase, greatly reducing time and personnel resources across the report's life cycle. This feature is discussed in detail in the "CODE GENERATION" section.

Note: The following screens and the data they contain have been altered to protect privileged information.

A brief 'walk-through' of the screens is given to help the reader understand system features and their use.

Display 1 Prototype Page Layout Screen

The system consists of three entry panels. The first (not displayed here) manages storage and retrieval of the prototype definition files. The second shown in Display 1 is the data entry screen which controls the general report layout. Page elements such as margins, width, and orientation, are entered here. The header and footer regions of the report are described and stored in separate files. This allows the use of a standard library of headers and footers which are shared by the project programming team. These features speed development and help establish a uniform look and feel to all reports in the project. The user can specify whether the prototype will use the data modeling feature and how many rows of 'dummy' data to display. Another point of realism is achieved through declaring page, division, and class break variables. These suppress the printing of the specified variables in non-break rows, print special character break-lines, and reset data modeling algorithms where used. The bottom section of the screen supports file maintenance on the header, footer, prototype, and SAS code libraries.

The third screen (a scrolling virtual panel) shown in Display 2. manages the specifics of the report definition (a coordinate grid has been added as a reference aid). Notice under CNTRL (A) the entries C, R, and H: options that indicate the report elements: columns, rows, and headers respectively. (The '+' sign indicates a definition which spans multiple lines.) Variable names are entered in (B). The system will calculate the optimal spacing of variables in the report based on an even distribution of unassigned or blank space. Or this spacing can be done manually by entries in WIDTH (C) and SKIP (G).
Cosmetic enhancements to the report are well supported. Vertical lines of any standard character can be placed after a variable on the report by an entry under BAR (H). Variable column titles are entered in (E). Multiple-line column titles use the ‘~’ character to indicate split points. Titles and data justification is handled by JUST (F) with the usual options of right, left, or center. Sub-headings that span multiple variables are supported by entering an ‘H’ in CNTL (A), and by indicating the ‘from’ and ‘to’ variables. These sub-heads can be underlined by placing a ‘U’ in the TYPE (D) field. If any variable titles are too long to fit horizontally within the defined WIDTH (C), they are automatically transposed to print vertically.

The system also has the capability to populate the prototype with over a variety of different types of metadata. To see examples of their use in the definition shown in Display 2 refer below to the list of coordinates:

**Meta-Data Type**
- Column (E) Rows:
  - Random Number Within Range 12, 42, 44, 46
  - Random Assignment from List 8, 23, 34-38
  - Assigned from List Ascending 2, 4, 6
  - Mandatory Repeating Groups 16-20

These are only 4 of 16 available data modeling options.
Pharmaceutical Industry

It may be helpful to spend a few moments cross referencing the screens (Displays 1 and 2) with the prototype output produced by the example definition they contain as shown in Output 1. If used appropriately the prototyper can create a model of a report which is virtually indistinguishable from the real thing; and do so in very short order.

CODE GENERATION

The prototype system has recorded all the attributes of the report in a form that is meaningful. Variable names, titles, data types, positions, even sort/break fields, are outlined in what is essentially a blueprint. This blueprint can now be 'built' into a SAS program by the code generator.

To be practical code generation must presuppose certain coding conventions and basic design assumptions. Hence the questions arose: What is the best way to structure a report program? What conventions should be adopted? These core assumptions determine every program the system produces. Model SAS code should be flexible enough to support a wide range of report styles. It should be easy to read, trace and correct problems, re-arrange layouts, or change data sources. It should have an efficient and open architecture. It was decided that a modular design best meets these goals, i.e. labeled subroutines and macros. The strategy for handling output is the DATA _NULL_ step. The code generator used the prototype definition in Displays 1 and 2 to produce several hundred lines of SAS code. Samples of some notable features of this code are presented as follows:

Note: Code was edited or omitted to preserve space.

The %COL macro is the control center for the layout of the report. Other macros reference %COL submitting the variable name as the argument, and receiving a numeric column position as the result. The variable column positions are calculated and exactly reproduce the layout viewed in the prototype definition. The column titles as well as the data are PUT in position by references to the %COL macro. This macro enhances the readability of the code because variable names are associated with every SAS PUT statement pertaining to a variable, e.g. column titles. In addition it documents all data items used in the report and their exact placement.

If it becomes necessary to alter a data item's position, in most cases, the programmer need only to change the column numbers in %COL.

The %PUTDATA macro writes the data lines on the report. However, not all data is controlled through this macro. Any division or class break fields will be PUT via the logic in the %REPORT macro. This macro controls only the data which will appear on every line of a tabular report. Notice the offsets and formats. These are derived automatically from the prototype definition (See Display 2. under (C), (D), (F), and (G)).
The %COLTTITLE macro is used to place column titles on the report. The titles are aligned with the justification, orientation, and split points assigned to them in the prototype definition (See Display 2. under (C), (E), and (F)). Also built automatically is the logic to support cosmetic features such as underlines, break-lines and vertical bars.

```macro
coltitle;

/* COLUMN TITLES ROW 1 OF 10 */
put @col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare2) 'l'
@col(efrac) 'e  "VENTRICULOGRAF"'
@col(s1) '+5 "DIAGNOSTIC ANGIOPH"'

/* COLUMN TITLES ROW 2 OF 10 */
put @col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare2) 'l'
@col(efrac) 'e 24 ' "'
@col(s1) '+24 ' "'

/* COLUMN TITLES ROW 3 OF 10 */
put @col(bare) 'l'
@col(bare) 'l'
@col(efrac) 'e  "E-MATRIX:"
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(efrac) 'e  A'
@col(bare) 'l'
@col(bare2) 'l'
@col(s1) ' "'

/* COLUMN TITLES ROW 4 OF 10 */
put @col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(efrac) 'e  " "
@col(bare) 'l'
@col(bare) 'l'
@col(s1) ' "'

/* COLUMN TITLES ROW 5 OF 10 */
put @col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(bare) 'l'
@col(s1) ' "'

NOTE: CODE FOR TITLE ROWS 6 THROUGH 10 OMITTED
```

The %REPORT macro invokes all the other macros in directing the production of the output. Data is sorted based on variables entered in the break options (See Display 1). The logic for page, division, and class variables are built in to support these break points. The file print parameters e.g. page size, line size, are calculated from the page layout section of the prototype definition. Headings and footnotes are controlled by subroutines which are called based on page breaks and/or the lines-left parameter relative to the size of the footnotes. In the head: subroutine date/time stamp information, the heading file data, and the %COLTTITLE macro are called on to write all headings and titles.

```macro
report(sb);

proc sort data= sb;
   by studynum treatment patient studyday;
data null;
   set in end=end;
   by studynum treatment patient studyday;
file print header=head notes= notes ll=ll
   page=n pagepos=pp
   lines=n linesleft=ls;
   if (n > 1 and (first.studynum or first.patient or ll <= 6))
      then do;
         link foot;
         put _page_;
         return;
      end;
   if top or first.patient or first.studyday then do;
      put @col(studyday) studyday 3. @;
      put @col(patient) patient 4. @;
      put amod(time) time 12. @;
      put amod(date) date 8. @;
      put amod(date) studynum 8. @;
      put amod(date) investigat 10. @;
      put amod(date) treatment group 10. @;
      put amod(date) center 20. @;
      put amod(date) study center 20. @;
      return;
   end;
put data;
if s0 then link foot;
return;
```

Even the best laid plans sometimes go astray. If then, it becomes necessary to change the design after the report is in production any code which has been added outside of the code generator can be salvaged by using an update feature. The user simply specifies a formerly generated program as the output for the code generator, alters the prototype definition to the revised design and types "U" (as opposed to "S" for normal code generation). Then the system will replace the macros which control the layout in the existing code without overwriting any hand-entered SAS.

It should be easy to see from these excerpts of the code the advantages of having the system sweat the details. Of course it is the rare application that can run right out of the box. It is still necessary to declare data files, write input statements, and do any special manipulations or merges of data sets that will contribute to the report. And, of course, there are limitations: This system in its current state would not be appropriate for primarily row-
Pharmaceutical Industry

driven report designs such as statistical summaries. Work is now being done on a version of the system in SAS/AF FRAMES. It is conceivable that aspects of these coding issues will be supported or enhanced.

CONCLUSION

Time and personnel resources can be leveraged by using CASE methods, such as prototyping and code generation, in SAS report development. The costs of front-loading the design process are well worth the benefits gained in the production side. Several dozens of SAS report programs have been developed using the system this paper describes. None had complications, design conflicts, or unwanted surprises. They all came out of production looking exactly like their prototypes. The aggravation associated with report cosmetic surgery had been virtually eliminated. The corresponding savings in terms of programmer time and sanity are substantial. It is hoped that this presentation will stimulate thinking on ways to further the use and scope of CASE technology in the production of SAS reports.

ACKNOWLEDGMENTS

Many useful suggestions and beta-test support were provided by the staff of the Woodbridge, NJ, and Blue Bell, PA, offices of Merck & Co., Inc.

Some of the macros and coding techniques were developed in whole or in part by members of the staff of BIOPHARM, Inc. of Blue Bell, PA. For more information refer to:


AUTHOR INFORMATION

System concept, design, and execution, as well as all presentation materials by:

W. Stetson Line
On-Line Consulting
530 Eighth Avenue
Warminster, PA 18974

(215) 442-1555