Structured Documentation for a Specific Type of Fourth Generation Language: Electronic Spreadsheets

CHARLES R. NECCO
NANCY W. TSAI

SCHOOL OF BUSINESS ADMINISTRATION
CALIFORNIA STATE UNIVERSITY, SACRAMENTO

BARBARA SMITH
INFORMATION ANALYST
COUNTY OF NAPA MANAGEMENT INFORMATION SERVICES

ABSTRACT

Many problems associated with using fourth generation languages stem from people and environments rather than the software itself. Some users become designers — resulting in application and sharing and recycling. If these applications are to be reliable, maintainable, and expandable, communication in this shared environment is critical. This paper explores the use of structured programming concepts, techniques, and tools used for solving design and documentation problems.

INTRODUCTION

Fourth generation languages have arrived, specifically in one of their more popular forms — electronic spreadsheets. During the past few years, the power of mainframe computers has become available to nontechnical users through desk top computers. Together, these software and hardware improvements have drastically reduced computing costs and gained the attention of potential business users in a multitude of problem-solving situations.

Because electronic spreadsheets are "user seductive," end users are allowed the capability of solving some of their own data processing problems by putting computers to work in various organizational settings. In some cases, however, after realizing the initial benefits of this easy to use software, electronic spreadsheet users have faced the reality of high cost end products lacking quality assurance, and difficult to maintain programs [10]. This situation emphasizes the inescapable need for policies and controls to manage the power unleashed by these fourth generation languages.

The data processing world has anticipated full arrival of fourth generation languages since their late 1970s introduction. The use of third generation languages, such as COBOL and FORTRAN, increased programmer productivity by replacing several assembly language instructions with one high-level language command. Organizations, unable to keep pace with growing demands for timely computer application development, recognized a need for higher productivity languages which would be user friendly, make applications easier to create/debug/maintain, and significantly speed up the application building process [7]. Such has not always been the case.

In the 1960s, third generation languages brought a new potential to develop application programs. A backlog of demands for modifications to existing programs and development of new applications soon evolved. Very little forethought was given to program design, and documentation was poor or nonexistent. Maintaining massive third generation language programs became a difficult task which continues to the present day.

Initially, the response to this situation focused on the immediate problems — programming and documentation. The solution centered around a structured approach which would be more logically clear and easier to communicate. The new methodology included programming tools and techniques such as top down design, modularization, structure charts, and Warnier-Orr diagrams [3,4].

Structured programming provided a means for solving many problems confronted in the third generation language environment. Since many structured techniques are procedure oriented, it is difficult to directly apply traditional structured programming techniques to nonprocedure-oriented spreadsheets. Instead, one must find different ways to achieve structured programming goals in the new electronic spreadsheet environment. The basis for meeting the goals of reliability, maintainability, and extendibility in spreadsheet applications begins with a well structured spreadsheet application design. But, even well structured applications may not intuitively be understood without supporting documentation communicating what is happening, where, and why. Its final documentation, then, must be structured to reflect both processes which guided the spreadsheet creation, as well as the structured nature of the spreadsheet itself.

The purpose of this paper is to explore the use of structured programming concepts, techniques, and tools used for solving design and documentation problems currently found in fourth generation language spreadsheet environments. A structured documentation methodology — specifically for declarative, nonprocedural spreadsheet languages — is presented. The documentation includes the spreadsheet map, the framework documents, the macro documents, the named range dictionary, and the user manual.

This structured documentation methodology utilizes the concepts of top-down design and modularity to facilitate understanding and communication. Among the documentation components the spreadsheet map, framework documents, and named range dictionary were developed to fit the unique documentation needs of a spreadsheet declarative language. The rationale for developing these new tools is provided, along with examples. Macro documents present the logic of individual macros using the Warnier-Orr diagramming technique, but other structured detailed logic diagramming techniques could be used. The user manual was written emphasizing the need for a readable and useful set of instructions from the user's perspective.

The authors, two MIS educators and a practitioner, are experienced in traditional and structured documentation techniques. In this research, an attempt was made to understand the requirements for documenting a spreadsheet language and then using or modifying existing tools and developing new tools as necessary. The methodology documented a Lotus 1-2-3 spreadsheet application which automated a quarterly claim report. The report was subsequently filed with the state of California by a county human services agency.

Electronic Spreadsheets

The first electronic spreadsheet package, VisiCalc, was developed in 1978 by a group of Harvard Business School students specifically for the financial analyses of business case assignments. Since then, spreadsheet software developers have used technological advances such as database operations and word processors to create a new generation of integrated spreadsheet packages with more capability and power. Spreadsheets do not require conventional computer programming; they do require an understanding of applied business mathematics, accounting, and financial modeling. A spreadsheet application can be as simple as totaling a series of columns or as complex as making and combining detailed financial projections.

Spreadsheet languages are not procedural programming languages, but rather a form of nonprocedural language called "declarative" [7]. In other words, a spreadsheet program is not a list of command statements executed sequentially from top to bottom. Instead, a spreadsheet is a framework of formulas and data organized into a matrix of rows and columns. Each row and column intersection is a cell where data or a formula can be stored. The command sequence is determined by cell relationships defined within the matrix framework. There may be multiple frameworks in a single spreadsheet. The spreadsheet applications are called templates. Keystrokes performing specific functions can be stored in macros.

Existing Documentation Standards

By and large, the spreadsheet market has been composed of non-data processing users, so it is not surprising that most spreadsheet-related literature has been written for this market. Until recently, the emphasis on spreadsheet package usage has been toward maximizing short-term benefits through the use of special features, functions, and applications. Minor attention was given to design and documentation considerations. The increasing realization of impending disaster and chaos — precipitated by this trend — has produced new concerns for improving the reliability and maintainability of spreadsheet applications.

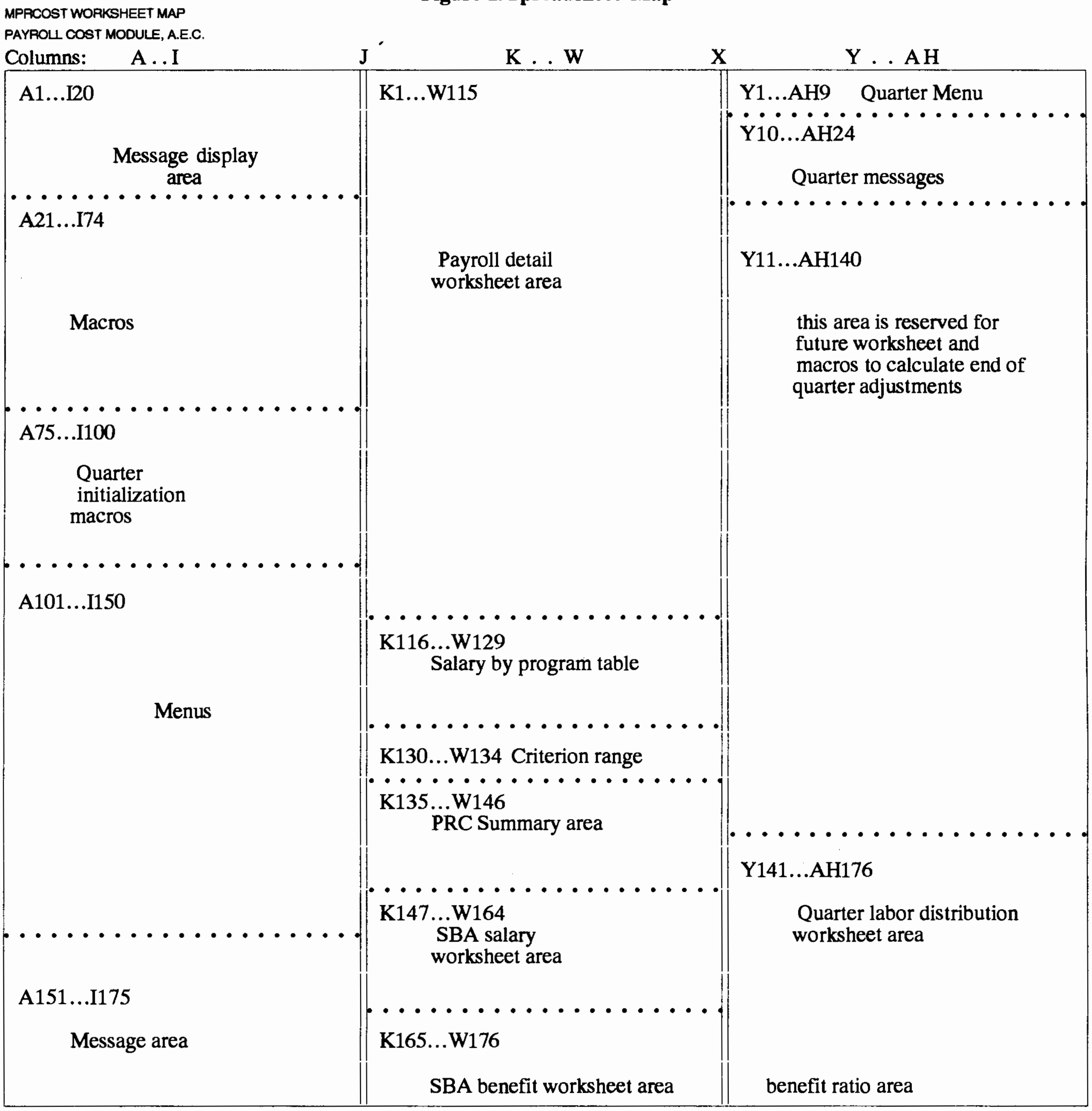
Early spreadsheet documentation, for the greatest part, was internal to the specific spreadsheet application and largely narrative; the first concern being to properly identify the spreadsheet and to explain its purpose and use. External documentation tended to be a spreadsheet printout with its supporting internal narratives. The addition of the macro capability functions brought concern about the spreadsheet's physical layout, where the macros and the frameworks should be placed relative to one another, and documentation of the cryptic macros themselves. Popular "how to" books stressed a need for internal macro documentation, recommending a style of storing comments next to each macro code line in the column to the right of the macro [6].

Not until recently have articles, directed at the non-data processing user, been published to specifically detail methods of improving spreadsheet design and documentation. Recommended design methods have emphasized consistency and clarity, structuring macros, and improved documentation. Significantly, the documentation suggested for both internal and external use has included non-narrative forms such as lists, tables, and charts. A Fritz Grupe authored article impressively offers 21 excellent tips for better spreadsheet documentation [5].

STRUCTURED DOCUMENTATION FOR ELECTRONIC SPREADSHEET APPLICATIONS

Five major structured documentation components for a

Figure 1. Spreadsheet Map



Source: Human Services Agency, Napa County, California

more reliable development — and subsequent maintenance— of spreadsheet packages are proposed: the spreadsheet map, framework documents, macro documents, the named range dictionary, and the user manual.

Spreadsheet Map

The recommended documentation begins by looking at the spreadsheet as a whole. The larger and more complex the spreadsheet, the more important the overview model becomes. The structured top-down approach should be adopted in this design stage [8]. The concept of using a graphic tool to decompose a complex system or problem into clear, logical, and interrelated component parts should be applied. This decomposition process should be continued until each part is a single human understandable function.

Therefore, it is important to emphasize the topography of the spreadsheet at the top level. In this context, topography refers to how the entire spreadsheet's various component parts lie relative to one another. The map shows the location of framework(s), macros, assumptions, etc. This spreadsheet map is an effective communication tool which provides the overall structure of the spreadsheet at a glance.

Each component of the map may be further broken down as deemed appropriate. Additionally, each component should be further documented. The map is both an overview and a table of contents for the spreadsheet and its external documentation. (See Figure 1.)

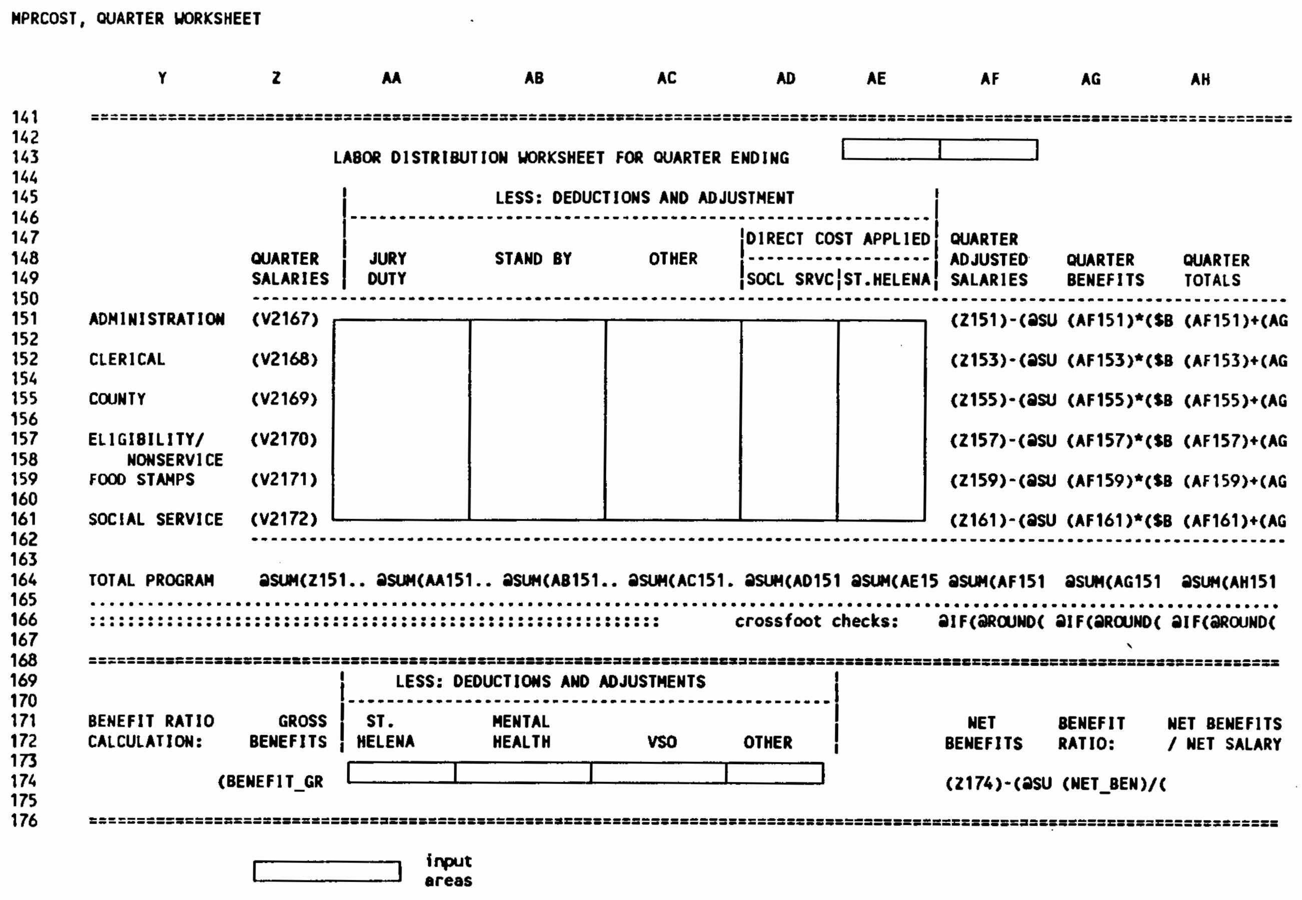
Framework Documents:

1. General. The framework is the single most important component of a spreadsheet. It is the root of the application, encompassing the model upon which the application is built. The documentation for each framework should include the framework matrix, input and formula tables, and a named range directory.

2. Framework Matrix. Much like the spreadsheet map, the framework matrix shows the overall layout of the framework. A printout, of the framework itself is necessary for the documents. On the printout, three areas need to be designated: title cells, input cells, and formula cells. Title cells are usually self-evident in the printout. Input and formula cells can easily be highlighted with colored marking pens for quick visual reference. (See Figure 2.)

3. Input and Formula Tables. To explain and define the model, both input and formula cells need further documentation. Many spreadsheet packages have a print option

Figure 2. Framework Matrix



```
MPRCOST, QUARTER WORKSHEET

            Y            Z          AA             AB             AC            AD         AE            AF           AG           AH

141   ===================================================================================================================================
142
143                                  LABOR DISTRIBUTION WORKSHEET FOR QUARTER ENDING
144
145                                |              LESS: DEDUCTIONS AND ADJUSTMENT              |
146                                |-----------------------------------------------------------|
147                                |                                       |DIRECT COST APPLIED| QUARTER
148                      QUARTER   | JURY         STAND BY       OTHER     |-------------------| ADJUSTED     QUARTER      QUARTER
149                      SALARIES  | DUTY                                  |SOCL SRVC|ST.HELENA| SALARIES     BENEFITS     TOTALS
150                      ----------------------------------------------------------------------------------------------------------
151   ADMINISTRATION     (V2167)                                                               (Z151)-(@SU (AF151)*($B (AF151)+(AG
152
152   CLERICAL           (V2168)                                                               (Z153)-(@SU (AF153)*($B (AF153)+(AG
154
155   COUNTY             (V2169)                                                               (Z155)-(@SU (AF155)*($B (AF155)+(AG
156
157   ELIGIBILITY/       (V2170)                                                               (Z157)-(@SU (AF157)*($B (AF157)+(AG
158       NONSERVICE
159   FOOD STAMPS        (V2171)                                                               (Z159)-(@SU (AF159)*($B (AF159)+(AG
160
161   SOCIAL SERVICE     (V2172)                                                               (Z161)-(@SU (AF161)*($B (AF161)+(AG
162                      ----------------------------------------------------------------------------------------------------------
163
164   TOTAL PROGRAM      @SUM(Z151.. @SUM(AA151.. @SUM(AB151.. @SUM(AC151. @SUM(AD151 @SUM(AE15 @SUM(AF151  @SUM(AG151  @SUM(AH151
165   .................................................................................................................................
166   :::::::::::::::::::::::::::::::::::::::::::::::::::::::::          crossfoot checks:    @IF(@ROUND( @IF(@ROUND( @IF(@ROUND(
167
168   =================================================================================================================================
169                                 |      LESS: DEDUCTIONS AND ADJUSTMENTS      |
170                                 |--------------------------------------------|
171   BENEFIT RATIO         GROSS   | ST.         MENTAL                         |          NET        BENEFIT      NET BENEFITS
172   CALCULATION:        BENEFITS  | HELENA      HEALTH         VSO     OTHER   |          BENEFITS   RATIO:       / NET SALARY
173
174               (BENEFIT_GR                                                               (Z174)-(@SU (NET_BEN)/(
175
176   =================================================================================================================================

                                input
                                areas
```

Source: Napa County, California

Figure 3. Input Table

Input Table: Quarter Worksheet

RANGE	DESCRIPTION
Row	
AA174 . . . AD174	benefit adjustments for direct costs, deductions = St. Helena, mental health, VSO, other format 99.999.99
Column	
	salary adjustments by program for direct costs, deductions format 99.999.99
AA151 . . . AA161	jury duty
AB151 . . . AB161	stand by
AC151 . . . AC161	other
AD151 . . . AD161	social service direct costs
AE151 . . . AE161	St. Helena direct costs
Cell	
AE143, AF143	unprotected for quarter date input from quarter macros no direct user input

Source: Napa County, California

which will list cell addresses and their contents. While this is an easy way to get a printed record of each cell, it is too disjointed to qualify as useful documentation. It is particularly inadequate in documenting formulas and models. The input and formula tables are a clear and concise way of doing this.

Most data input is organized into records (rows) and fields (cells). The input table lists each input field, its cell range, data source, and brief description of the data item. (Figure 3 illustrates an input table which corresponds to input areas defined in Figure 2.)

Formula tables are especially critical because formulas are the building blocks of an application model. A formula which is altered accidentally or copied incorrectly can sabotage the best model. Many formulas are too long to be displayed in full on the framework map printout but need to be listed completely in supporting documentation.

The formula table lists each base formula, its cell range, and a brief description and/or explanation of its function. If a formula is copied over a range of cells, it should be noted whether cell references in the formula are absolute or relative. The descriptive narrative can include the formula in conventional mathematical notation, how formula results should look, result acceptance criteria, etc. Complex formulas should be explained, including what is being calculated, formula components, and mathematic or model source references outside the spreadsheet. (See Figure 4.)

Named Range Directory (framework)

It is important to keep track of named ranges within a framework, especially if they are referenced elsewhere in the spreadsheet. The named range directory lists ranges and cell addresses for each framework. Named ranges are further documented in the spreadsheet's named range dictionary. (See Figure 5.)

Figure 4. Formula Table

Formula Table: Quarter Worksheet

Range	**Formula**	**Description**
Z151 . . . Z161	(range name)	program gross salaries from the salary by program table
AF151 . . . AF161	(Z151) - (@SUM (AA151 . . . AE151))	all relative reference subtracts row entries for columns AA through AE (deductions) from quarter gross salary in column Z
AG151 . . . AG161	(AF151) * ($BEN - RATIO)	(AF151) relative reference (BEN - RATIO) absolute reference multiplies BEN - RATIO (range name for AG174) times quarter adjusted salary in column AF
AH151 . . . AH161	(AF151 + AG151)	all relative reference totals row entries, columns AF. AG
Z164 . . . AH164	@SUM (Z151 . . . Z161)	all relative reference adds column, rows 151 through 161
Z174	(BENEFIT - GROSS)	range name for V175 gross benefits for quarter
AF174	(Z174) - (@SUM (AA174 . . . AE174))	subtracts columns AA through AD (deductions) from gross benefits in column Z
AG174	(NET - BEN) / (NET - SAL)	calculates benefit ratio by dividing net benefits by net salaries (uses range names)
Crossfoot		
AF166	@IF (@ROUND (NET - SAL,2) = @ROUND (Z164) - (@SUM AA164 . . . AE164),2), + Z164 - @SUM (AA164 . . . AE164), @ERR	
AG166	@IF (@ROUND (AH164,2) = @ROUND (NET - BEN,2), (NET - BEN), @ERR	
AH166	@IF (@ROUND (AH164,2) = @ROUND (+NET - SAL + AG164,2), (+NET - SAL + AG164), @ERR These are crossfooting error check formulas. If the column total is equal to the relative formula of the column, the relative column formula for the row is displayed in this cell If they are NOT equal, display "Err." Rounding to two decimal places is used to prevent trailing decimal.	

Source: Napa County, California

Figure 5. Named Range Directory

MPRCOST WORKSHEET
PAYROLL COST MODULE. A. E. C.

RANGE NAME DIRECTORY (ALPHABETICAL)

Range Name	Address	Worksheet
AD - GROSS	V119	TABLE
BENEFIT - GROSS	V175	SBA
BEN - RATIO	AG174	OTR
BLANK	A10	
CL - GROSS	V120	TABLE
CODE - INPUT - CELL	L133	CRITERIA
CORRECT - MENU	B137 . . . E138	
CO - GROSS	V121	TABLE
DATE1	E7	
DATE2	F7	
DATE - CELL1	S6	PRDTL
DATE - CELL2	T6	PRDTL
DATE - MENU	B88 . . . E89	
DRIVER - MENU	B102 . . . E103	
ENTER - MENU	B119 . . . H120	
EN - GROSS	V122	TABLE
FS - GROSS	V123	TABLE

Source: Napa County, California

Macro Documents:

1. General. Macros make more efficient use of the programming language of electronic spreadsheets. Developing macros is like creating your own function keys to customize a spreadsheet application. This is done by storing — usually in spreadsheet cells — a series of keystrokes in the spreadsheet package's command language. The macro cell or cells are given a range name representing a unique key combination — as the keys are invoked, keystrokes stored in the name cell begin to execute in sequence exactly as stored. Most contemporary spreadsheet packages have predefined powerful macro function commands which allow branching, limited if-conditioning, calls to other macros, and menu building.

If one is not well-acquainted with a spreadsheet package's command language, reading the stored macros is difficult — if not impossible. The design and documentation of these highly developed macros is akin to that required for third generation program language applications. External documentation of these macros becomes increasingly critical for application evaluation and maintenance.

Top-down documentation of macros means documenting encompassed procedures from the general to the specific. Two tools for top-down documentation can be used: Warnier-Orr diagrams for general documentation of structured macros, and a macro dictionary to document macro detains [11,12].

2. Warnier-Orr Diagrams. For complex macros, the Warnier-Orr structure gives a quick overview of the macro

Table 6. Warnier-Orr Diagram for a Menu Macro

		/ruWORKSHEET~ /reWORKSHEET~ /ruDATE~ /ruUNIT~	
	SOCIAL SERVICE Load Social Service recap worksheet	/ruSPVSRHRS~ /ruNAMES~ /ruINPUT~ {goto}A1~ /fcceTSRSW~ {goto}B1~ /wtv	
		/xg\M~ ________	\xmMAINMENU~
		/ruWORKSHEET~ /reWORKSHEET~ /rpWORKSHEET~ /ruDATE~ /ruUNIT~	
	ELIGIBILITY Load Eligibility and Nonservice	/ruSPVSRHRS~ /ruNAMES~ /ruINPUT~	
LOADMENU	recap worksheet	{goto}A1~ /fcceTSRSW~ {goto}B1~ /wtv	
		/xg\M~ ________	\xmMAINMENU~
		/ruWORKSHEET~ /reWORKSHEET~ /rpWORKSHEET~ /ruDATE~ /ruUNIT~	
	FOOD STAMPS Load Food Stamps recap worksheet	/ruSPVSRHRS~ /ruNAMES~ /ruINPUT~	
		{goto}A1~ /fcceTSRSW~ {goto}B1~ /wtv	
		/xg\M~ ________	\xmMAINMENU~
	QUIT		
	Return to main menu	/xg\M~ ________	\xmMAINMENU~

Source: Napa County, California

and/or menu relationships and serves as a table of contents for the detailed macro documentation. In simpler cases, the diagrams may be carried out to the macro code level providing all external documentation is in one format. (See Table 6.)

3. Macro Dictionary. For understandability, complex macros need to be broken down into digestible phrases. Sometimes macros — especially those called by menu selection — perform a series of related tasks. Within each individual macro documentation, the section of codes accomplishing specific tasks should be separated and tasks should be identified.

Each macro dictionary document should begin with the

Figure 7. Macro Dictionary

Macro Dictionary				
System: Module: File Name:		HSDS ADMINISTRATIVE CLAIM PAYROLL COST MPRCOST	Date Prepared: Prepared by:	03/86 BAS
Reference:		(MAIN) DRIVER.QUARTER MENU.	PRINT REPORT	
Purpose:		To print quarter labor distribution report.		
Working				
Name	Command	Description		
PRINT REPORT		(Print the quarter labor distribution worksheet)		
This macro performs the following tasks: (1) Copies a print message to the user. (2) Puts the macro on "hold" until the user has prepared the printer and is ready to print. (3) Prints the worksheet. (4) Copies a new message to the user and returns to the main menu.				
	/wtc		worksheet, title, clear - clears any title lock	
	{goto} {home}~		move cursor to home position	
	/reMESSAGE - AREA~		erase message box	
	/cQUARTER - PRINTMSG~MESSAGE - DISPLAY~		copy print message to the message display area	

Source: Napa County, California

macro's name of menu selection, brief statement of purpose, and key identity stroke labels/range defining it. Subroutines unique to macros may also be included, but should be identified in the name and label sections. Organization of the macro dictionary will vary with a macro's structure, but a rule of thumb is to group together menu selections before moving to the next menu level. (See Figure 7.)

Names Range Dictionary (spreadsheets)

Named ranges in spreadsheets are equivalent to data or variable names in procedural programming languages. They are used in commands, formulas, and macros. In declarative spreadsheet languages, named ranges cannot be considered data flowed, data stores, or data processes. Depending upon use, named ranges may exhibit characteristics of any combination, or none of them. Clearly, traditional data dictionary formats will not suffice. A new form should be used that will give exact and complete definitions for each named range including where the named range is referenced.

Therefore, the named range dictionary should include basic information regarding system, spreadsheet and/or framework, and by whom and when the dictionary entry was prepared. The dictionary should list the range name or its alias, the cell location, named subset ranges, a brief named range description, and reference location. (See Figure 8.)

The completed named range dictionary shows relationships between frameworks, formulas, and macros. When applications must be modified, the dictionary provides a map to evaluate the possible "ripple effects" of a change which might otherwise go unnoticed. Therefore, it provides clear and concise documentation for the required maintenance to a spreadsheet application.

Figure 8. Named Range Dictionary

System:	HSDS ADMINISTRATIVE CLAIM	**Date Prepared:**	03/86
Module:	PAYROLL COST	**Prepared by:**	BAS
File Name:	MPRCOST		
Range Name:	AD - GROSS		
Alias:	(Z151)	**Location:**	V119
Contained in:	PRC - TABLE		
DESCRIPTION:			
CELL FOR TOTAL ADMINISTRATIVE (AD) GROSS SALARIES FOR QUARTER			
WHERE USED:			
FORMULA TABLE: QUARTER WORKSHEET (Z151)			

Source: Napa County California

User Manual

Often the most important, but least considered component of external documentation for a spreadsheet application is the user manual. If anyone other than the designer is to use the spreadsheet, there must be coherent instructions. Even when the application is menu driven or otherwise "user friendly," there remains an important need for clear, easy to follow, written support to facilitate the spreadsheet application's use.

The user manual can become critical to the success or failure of an application; a poorly written manual can effectively sabotage the user acceptance of a new application. On the other hand, a well written manual can speed acceptance and increase user productivity. The user manual should be informative rather than technical. It is the communication link between user and spreadsheet application. The important point in user documentation is to make the manual clear, concise, direct — and most importantly, usable.

The user manual should include a simple description of the spreadsheet application's purpose, including what it does, and its inputs and outputs. Assumptions about the user's expertise should be stated at the manual's beginning and referenced to locations of supporting detail for any "assumed" information. The manual's main body should contain instructions on operating the application.

The following guidelines should be considered while preparing the user manual: For clear and easy to follow instructions, use numbered step-by-step instructions whenever possible, minimizing cross-references [2]. Show users — rather than tell them. In other words, emphasize instructions and minimize descriptive narratives — the user should be doing, not reading. Use illustrations: screens/diagrams/pictures. They are truly worth the proverbial thousand words and can greatly enhance and clarify instructions.

Layout or presentation of material is also important. Wide margins, lots of blank "white" space, and an easy-on-the-eyes typeface will give the manual an uncluttered, easy to read look. Another suggestion is adding guideposts for readability. These include a table of contents, and index, and tabs to identify sections. Placed in a three-ring binder, such a manual is attractive, easy to use, and conveniently updated.

One final and extremely important guideline is to test the manual before its publication. Let some typical users try using drafts of the manual. Their feedback is an invaluable measure of the manual's strengths, weaknesses, and probable success. With this additional information, the manual can be improved from the user perspective so that it, and the application, will be successfully used and accepted. (Table 9 illustrates a portion of a completed user manual.)

CONCLUSIONS

Fourth generation languages have generated a great deal of interest and enthusiasm, but in some respects they have not measured up to the high expectations created by their advance notices. Many problems associated with using fourth generation languages stem from people and environments rather than the software itself. These high-productivity languages still require a need for logic clarity in application designs and suitable documentation of applications. A need for precise communication is intensified in the new shared

Figure 9. User Manual

USING THE PAYROLL COST SPREADSHEET

The newly created spreadsheet uses a DRIVER MENU to lead the user through a series of steps to enter data, print, and save the worksheet. When in the "ready" mode, this menu can always be accessed by holding down the "Alt" key and pressing D. Whenever the spreadsheet file is retrieved, this Driver menu will appear at the top of the screen.

The menu selections are:

MONTH MENU:	Month activities -- enter PRC or SBA data -- sort records -- print worksheets, etc.
QUARTER MENU:	Quarter activities
SAVE:	Save the worksheet to a disk file
EXIT:	End Lotus 1-2-3 session

You may choose a selection in two ways:

(1)	Move the cursor to the chosen selection and press return,
	or,
(2)	Type in the first letter of your selection choice.

MONTH and QUARTER selections present new menus. SAVE returns to the Driver menu after saving the worksheet. EXIT gives the choice whether to quit or to continue.

Source: Napa County, California

environments.

The documentation methodology presented in this paper was tested with a major payroll cost system's spreadsheet application in generating a government agency's quarterly claims report. The following conclusions were made following the research and actual experience of application development:

(1) Evidence that some fourth generation language applications are following the same path of third generation languages: poorly designed, unreliable, hard-to-maintain codes. It seems that little has been learned from third generation language experiences and the trauma must be relived to reinvent solutions. Three reasons this situation may develop: (a) unrealistic expectations about fourth generation languages, (b) structured tools often do not apply directly to fourth generation languages, and (c) fourth generation languages have attracted a new group of non-data processing users who are not aware of structured techniques.

(2) The concepts of structured approach still apply in the fourth generation language environment. An increasing popularity of "user-friendly" software leads to more users, more applications, and more application designers. Many users become designers, and there is an increasing incidence of application sharing and recycling among user-designers. If these applications are to be reliable, maintainable, and expandable, then communication in this shared environment is critical. More than ever, there is a need to educate and train users in the structured approach to application building and documentation.

(3) The declarative nature of spreadsheet languages does require a modification of the use of structured tools for application documentation. The model which is expressed in

a framework cell relationship using formulas, macros, and named ranges is unique to these packages and must be specifically addressed in verifying and maintaining application templates.

(4) The proposed documentation tools are tedious and time consuming. They do provide essential information and specifically answer a need for spreadsheet documentation, but they also add considerable time to the application building process. Without an accompanying enforced documentation standard policy, most designers would probably avoid the development of necessary documentation altogether.

(5) In this specific application, the results to date suggest that the user manual has been important for training new employees and answering processing questions necessary for generating reports. The overall documentation has been considered a timesaver whenever there is a need to revise the system for changes in information requirements. Therefore, the authors believe that the documentation tools presented in this paper are an improvement over existing methods for documenting spreadsheets because they specifically consider the unique form and relationships of commands in a spreadsheet declarative language. This proposed documentation methodology is not the definitive answer to spreadsheet documentation, but it should be considered by those organizations who desire improved procedures for developing and documenting electronic spreadsheet applications.

REFERENCES

[1] Berry, T., "Writing Structured Macros," *Lotus,* 1, Number 4, August 1985, pp.64-67.

[2] Blake, G. and Bly, R.W., "Ten Tips for Better User Manuals," *Computer Decisions,* 16, Number 11, September 1984, pp.68-70.

[3] DeMarco, T., *Structured Analysis and System Specification,* Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[4] Gane, C. and Sarson, T., *Structured System Analysis: Tool and Techniques,* Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[5] Grupe, F., "Tips for Better Worksheet Documentation," *Lotus,* 1, Number 4, August 1985, pp.68-69.

[6] Howitt, D., "Avoiding Bottom-Line Disaster," *InfoWorld,* 7, Number 7, February 11, 1985, pp.26-30.

[7] Martin, J., *Fourth Generation Languages, Volume One, Principles,* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[8] Martin, J. and McClure, C., *Diagramming Technique for Analysts and Programmers,* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[9] LeBlond, G. and Cobb, D., *Using 1-2-3,* Que Corporation, Indianapolis, Indiana, 1983.

[10] Necco, C., Gordon, C. and Tsai, N., "Fourth Generation Languages and Microcomputers," *The Journal of Computer Information Systems,* Winter 1987.

[11] Orr, K., *Structured Systems Development,* Yourdon Press, New York, 1977.

[12] Orr, K., *Structured Requirements Definition,* Orr & Associates, Topeka, Kansas, 1981.

[13] Smith, B., "Structured Techniques Applied to Fourth Generation Languages," Master's Project, California State University, Sacramento, California, Spring 1986.

ABOUT THE AUTHORS

Charles R. Necco *is Professor of Management Information Systems in the School of Business Administration at California State University, Sacramento. He received his Ph.D. in business from the University of Illinois. His major teaching and research interests are in the areas of systems analysis and design, fourth generation languages, and decision support systems. His work has been published in* MIS Quarterly, *the* Journal of Information Systems, *the* Journal of Systems Management, *and others.*

Nancy W. Tsai *received her Ph.D. in business from the University of Texas at Austin. She is presently a professor of Management Information Systems in the School of Business Administration at California State University, Sacramento. Her teaching and research interests are related to database systems, fourth generation languages, and systems analysis and design. She has published papers in several journals such as* MIS Quarterly, *the* Journal of Systems Management, *and the* Journal of Computer Information Systems.

Barbara Smith *is an Information Analyst with the County of Napa Management Information Services Department. She also develops training curriculum and teaches Computer Science and MIS courses at Napa Valley College.*