A generalized methodology for spreadsheet applications (GMSA)

Jack T. Marchewka  
*Northern Illinois University*

Joseph L. Sessum  
*Kennesaw State University*

Follow this and additional works at: [https://scholarworks.lib.csusb.edu/jiim](https://scholarworks.lib.csusb.edu/jiim)

Part of the Management Information Systems Commons

**Recommended Citation**  
Available at: [https://scholarworks.lib.csusb.edu/jiim/vol4/iss2/4](https://scholarworks.lib.csusb.edu/jiim/vol4/iss2/4)

This Article is brought to you for free and open access by CSUSB ScholarWorks. It has been accepted for inclusion in Journal of International Information Management by an authorized editor of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
A generalized methodology for spreadsheet applications (GMSA)

Jack T. Marchewka  
Northern Illinois University

Joseph L. Sessum  
Kennesaw State College

ABSTRACT

Most information system methodologies are intended to be used by information systems (IS) professionals. However, few tools and methodologies have been proposed and developed for the non-IS professional. The methodology presented in this paper is designed to provide the non-IS professional with a set of tools to take advantage of the rich features associated with present spreadsheet software packages. Moreover, the methodology presented in this paper is intended to serve as an education and training device to aid in the identification and subsequent solution of general business problems without requiring total reliance on the features of any specific spreadsheet software package or operating system environment. A methodology for developing spreadsheet applications offers significant productivity benefits because of the preliminary efforts directed at the identification of the key elements that influence the situation being modeled and the associated reduction in trial-and-error attempts during application development.

INTRODUCTION

The electronic spreadsheet market is worth approximately $1 billion (Wall Street Journal, 1993). The growth in electronic spreadsheets can be attributed mainly to their ease of use, and, more importantly, the ability for end users to develop and maintain their own applications. The widespread use of electronic spreadsheets has contributed to the growth of end user computing. In fact, a survey of Fortune 500 firms in 1991 suggests that 60 to 80 percent of information technology (IT) budgets are being spent on end user computing (Amoroso & Cheney, 1991). Even though spreadsheet-based applications are generally less data intensive than their database counterparts, their importance in business situations are equally as important in both tactical and strategic operations.
Electronic spreadsheet packages provide a great deal of flexibility by allowing the "user-developer" to readily make changes to the spreadsheet application. Given this flexibility, one may wonder whether a methodology for developing spreadsheet applications is necessary or even desired. It appears, however, that spreadsheet applications are too often the result of trial and error methods which present the results without readily discernible paths of the underlying rationale. Although few studies have considered the quality of the user developed spreadsheets, copious amounts of anecdotal evidence suggests that spreadsheet applications are error-prone and difficult to manage and maintain (Benham, Delaney, & Luzi, 1993). The few studies that provide some empirical evidence suggest further support. For example, Roberts (1980) reports that 80 percent of the spreadsheets that were audited contained at least one error. In addition, Ronen, Palley, and Lucas (1989) point out that many spreadsheet applications developed by users have logic mistakes, unreliable and/or incomprehensible output, and are difficult to maintain and manage. One reason for these problems may be that user-developers use prior analytical techniques in creating the spreadsheet application (Alavi & Weiss, 1986).

The use of a methodology for developing spreadsheet applications has several advantages. For example, each piece of the methodology provides a pathway that leads to a desired end result. Here the user-developer may understand more fully the underlying model and identify all of the variables at the outset before the model is represented using the spreadsheet software package. In addition, the user-developer can highlight and make explicit the assumptions behind the spreadsheet application. More time invested in understanding the problem "up front" may allow the user-developer to waste less time making unnecessary changes to the spreadsheet application. Furthermore, using a methodology provides a set of documentation for each spreadsheet application (Plane, 1994).

The remainder of this paper includes four main sections. The first section provides a general overview of some previously proposed methodologies for developing spreadsheet applications. In the next section, we propose a methodology that attempts to incorporate some of the rich features associated with present spreadsheet software packages. The following section provides two examples to demonstrate the methodology and its flexibility for designing spreadsheet applications. The last section of this paper provides a summary of the proposed methodology and directions for future development.

BACKGROUND

The salient processes of the analysis and design phases of the formal life cycle of a general business system are well defined, especially for database oriented systems. Unfortunately, there have been only a few attempts to parallel applications that may be shorter-lived and more immediately needed than spreadsheet applications.

Early attempts to develop spreadsheet analysis and design tools and techniques have tried to adapt traditional structured programming techniques. For example, Ronen, Palley, and Lucas (1989) adapt a data flow diagram approach for developing spreadsheet applications. Since
most spreadsheet applications consist of developing models and testing assumptions, a structured tool that focuses on the flow and transformation of data may not be practical for most spreadsheet applications (Benham, Delaney, & Luzi, 1993).

To overcome this problem, Benham, Delaney, and Luzi (1993) present a structured technique for developing spreadsheet applications that mirrors the sequence or phases of the traditional system development life cycle. The authors suggest that the design of a spreadsheet application should include, at a minimum, five sections. First, an introductory section should include any necessary documentation that outlines the purpose of the spreadsheet, and any directions or descriptions on how to use the spreadsheet or its underlying model and organization. The second section should include all of the data and assumptions used in the spreadsheet. Subsequently, the third section includes the model underlying the spreadsheet application. Only formulas are to be included in this section. The analysis section makes up the fourth section. Here the outcome or results to be printed are included. The last section contains all of the macros to provide a convenient section for documentation, testing, and maintenance.

The approach presented by Benham, Delaney, and Luzi (1993) provides spreadsheet developers with a structured approach for creating and documenting their applications. Moreover, this approach may be quite useful for enforcing standards among all spreadsheet applications. However, many of the "good programming" practices that are to be incorporated into the design of the spreadsheet application are implied. More specifically, users who develop their own spreadsheet applications may still have problems in logic or create applications that are unreliable or are difficult to maintain. What is needed is a tool to first help these user-developers understand the underlying model, the data and assumptions needed before sitting down before the computer.

In attempt to overcome this limitation, Plane (1994) suggests that an abstract model is needed to help the user-developer visualize the real-world problem or situation. He suggests using an influence model because the user-developer is better able to understand the logic of the model before attempting to create the spreadsheet application. For example, the user-developer may be able to see not only the direct influences on certain variables, but the indirect influences as well. Figure 1 provides an example of the direct and indirect influences on total costs. Although total costs are directly influenced by fixed and variable costs, they are influenced indirectly by the cost per unit, the number of units produced, rent, and insurance. Any variable that does not have an arrow pointing to it is considered to be data or decision variables that are controlled by the decision maker.

Continuing with Plane's (1994) approach, the next step is to document the relationships outlined in the influence model. These relationships are expressed in terms of the variables and formulas that are to be included in the spreadsheet application. The variables and formulas outlined in the documentation are then easily entered into the spreadsheet package using cell formulas. Table 1 provides an example of all of the variables and formulas that represent the relationships in Figure 1.
Figure 1. Influence Diagram Example

Total Costs = Variable Costs + Fixed Costs

Variable Costs = Cost Per Unit * Number of Units

Fixed Costs = Rent + Insurance
One disadvantage to Plane's (1994) approach is that it does not take advantage of the spreadsheet package's rich features. In addition, an interim step may be needed so that the user-developer can more easily make the transition from the abstract model to implementation of model using the spreadsheet software package. This interim step should allow the user to waste less time making unnecessary changes to the spreadsheet in "trying to get things to look just right."

We now present a methodology for developing spreadsheet applications that includes an integrated set of tools. These tools can be linked in such a way as to provide the user with a methodology that begins with an abstract model and allows the user-developer to end with an application developed on any spreadsheet software package.

A GENERALIZED METHODOLOGY FOR SPREADSHEET APPLICATIONS (GMSA)

We begin by summarizing the methodology proposed in this paper. The user-developer first creates an abstract model of a given problem or situation that is to be analyzed, implemented, and presented using a spreadsheet software package. By first developing an abstraction of the problem or situation, the user-developer should be better able to identify the assumptions and relationships to be included in the spreadsheet application.

The tools used to create an abstract model are based upon an influence diagram but have been tailored to take advantage of the rich features associated with electronic spreadsheets. An example of a minimal set of symbols that can be used in the influence diagram is suggested in Figure 2.

Figure 2. Extended Influence Diagram

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rectangle</td>
<td>Input Variable</td>
</tr>
<tr>
<td>Diamond</td>
<td>&quot;IF&quot; Statement</td>
</tr>
<tr>
<td>Arrow</td>
<td>Direction of Influence</td>
</tr>
<tr>
<td>Circle</td>
<td>Value/Calculation</td>
</tr>
<tr>
<td>Double Diamond</td>
<td>Lookup</td>
</tr>
</tbody>
</table>
The circle in Figure 2 can be used to represent a variable that is either a fixed or calculated value. A rectangle or square represents an input variable that may change as the result of a change in the assumptions of the model. A diamond indicates an "if" or conditional statement or condition. The diamond within a square is an example of symbology used to represent the more rich, robust, and distinctive spreadsheet features and capabilities. In this case, the symbol is used to represent the table lookup (vertical or horizontal) features.

As mentioned earlier, once the user-developer creates an abstract model, the salient variables and their relationships are readily identified. The user-developer is thus provided with some assurance that the model is descriptive and the relationships are explicit. The next step then requires the user-developer to identify the variables, formulas, and their relationships. At this point the user-developer would actually map out the variables and associated relationships by writing out the formulas and/or other equations to be used in the model. The third step in this scheme is particularly important because it is often the first step that new initiates to spreadsheet application development attempt to take. This step requires the translation of the abstract model into a layout form that will resemble the spreadsheet to be implemented (Morgan, 1993). Here the user-developer visualizes the layout of the spreadsheet before actually implementing the model in the selected spreadsheet software package. The actual formatting of labels and values and specifics, such as justification, number of decimal places, currency, dates, etc., map how the final spreadsheet should appear. After the layout form is completed, the user-developer can then implement the layout form using any spreadsheet software package.

Each step is now described in more detail. Two examples are provided to demonstrate the application of the methodology.

APPLICATION OF GMSA

Three examples are provided in this section to demonstrate the flexibility and richness of the methodology. Figure 3 provides an example of a simple model that includes a number of variables (and their relationships) to compute profit.

By using a set of symbols, the relationships within the model become richer. For example, Advertising and Unit Price are represented within rectangles. Since rectangles represent input variables, the representation of the model in Figure 3 suggests that Advertising and Unit Price are under the user-developer's (or organization's) control and thus can be changed. Since Unit Cost and Fixed Costs are represented in the model using circles instead of rectangles, the model in Figure 3 suggests that the user-developer or the organization cannot change or manipulate these variables directly. If, for example, the cost per unit or the fixed costs could be changed or controlled directly, then a rectangle would be more appropriate. By using a set of symbols, different assumptions can be made more explicit. In addition, one can easily see that the number of units sold is directly influenced by the level of advertising; however, the amount spent on advertising also has an indirect effect on Revenue, Variable Costs, Fixed Costs, and, ultimately, the amount of Profit made by the firm.
Once the abstract model is completed, the next step is to identify the values and formulas used to compute Profit. In general, the direction of influence flows from left to right; however, the formulas and values may be identified in either direction. For example, Table 2 provides the variables and formulas that are associated with the model in Figure 3.

Table 2. Document Table

<table>
<thead>
<tr>
<th>Market Demand</th>
<th>Unit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units Sold</td>
<td>Unit Price</td>
</tr>
<tr>
<td>Fixed Costs</td>
<td></td>
</tr>
</tbody>
</table>

Units Sold = 20% * Market Demand  
Variable Costs = Unit Cost * Units Sold  
Revenue = Units Sold * Unit Price  
Total Cost = Fixed Costs + Variable Costs  
Before Tax Profit = Revenue - Total Costs
After the formulas and values have been identified, the user-developer then creates a layout form. The layout form provides a bridge between the abstract model and the actual implementation of the application. An example of a layout form is provided in Figure 4.

**Figure 4. Example 1 Layout Form**

<table>
<thead>
<tr>
<th>ABC, Inc.</th>
<th>Market Demand</th>
<th>Unit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit Model</td>
<td>$999,999.99</td>
<td>$9.99</td>
</tr>
<tr>
<td>Unit Price</td>
<td>$9.99</td>
<td>$999,999</td>
</tr>
<tr>
<td>Revenue</td>
<td>$9,999,999.99</td>
<td></td>
</tr>
<tr>
<td>Fixed Cost</td>
<td>$999,999.99</td>
<td></td>
</tr>
<tr>
<td>Variable Cost</td>
<td>$999,999.99</td>
<td></td>
</tr>
<tr>
<td>Total Costs</td>
<td>$9,999,999.99</td>
<td></td>
</tr>
<tr>
<td>Before Tax Profit</td>
<td>$9,999,999.99</td>
<td></td>
</tr>
</tbody>
</table>

The purpose of the layout form is to help the user-developer visualize what the spreadsheet will look like and will include any formatting and titles. Upon completion of the layout form, the user-developer then implements the form using a spreadsheet software package. The layout form provides a map in terms of the cell entries for labels, values, and their formats. For example, the value associated with Revenue would be formatted using currency with two decimal places, while Units Sold would be formatted using the fixed format with no decimal places.

Our experience using this methodology suggests that input variables (i.e., variables that are represented using rectangles in the abstract model) should be included in a separate section of the spreadsheet. The user-developer may choose to highlight these variables in the layout form as assumptions that impact the model. In the actual implementation of the model using the spreadsheet software package, absolute references to these cell values may be incorporated into the cell formulas. For example, since the level of advertising and the price per unit may change, the user-developer may desire to test a number of "what-if" scenarios. By using absolute cell
references in the cell formulas, the user-developer can change the assumptions and see the impact of those changes. This may reduce the potential for error since the user-developer changes only the assumptions and not the underlying formulas. Variables that are assumed not to change (i.e., represented in the abstract model using circles), such as fixed costs and cost per unit, should be cell-protected since these values are assumed to be outside the control of the user-developer. These ideas can be easily annotated and subsequently documented on the layout form.

Figure 5 provides an example of an abstract model that uses a table lookup to determine students' grades. The rectangle boxes for the exam and project weights indicate that these are all input variables that may change and allow the user-developer to perform what-if calculations. On the other hand, grades could be cell-protected since these values should not change.

Figure 5. Example with Table Lookup Function
Table 3 provides the variables and formulas associated with the abstract model in Figure 5. Figure 6 provides the layout form that would be implemented using any spreadsheet software package.

### Table 3. Document Table

<table>
<thead>
<tr>
<th>Exam 1 Weight</th>
<th>Exam 2 Weight</th>
<th>Exam 3 Weight</th>
<th>Project Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Grade = Exam 1 * Exam 1 Weight + Exam 2 * Exam 2 Weight + Exam 3 * Exam 3 Weight + Project * Project Weight</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Final Grade**: A if $90 \leq \text{Average Grade} \leq 100$
B if $80 \leq \text{Average Grade} < 90$
C if $70 \leq \text{Average Grade} < 80$
D if $60 \leq \text{Average Grade} < 70$
F if $0 \leq \text{Average Grade} < 60$

### Figure 6. Example 2 Layout Form

#### Final Grades
Spring Semester
BUS 101

<table>
<thead>
<tr>
<th>Exam 1 Weight</th>
<th>Exam 2 Weight</th>
<th>Exam 3 Weight</th>
<th>Project Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>99%</td>
<td>99%</td>
<td>99%</td>
<td>99%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
<th>Exam 1</th>
<th>Exam 2</th>
<th>Exam 3</th>
<th>Project</th>
<th>Ave</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name (1)</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>X(1)</td>
</tr>
<tr>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Name (25)</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>999</td>
<td>X(25)</td>
</tr>
</tbody>
</table>
The purpose of this paper was to introduce a generalized methodology for developing business-oriented electronic spreadsheet applications. The methodology consists of four steps:

1. Create an abstract model to identify the salient values, variables, calculations, and directions of influence.
2. Using the abstract model, document all values, variables, and formulas in a table.
3. Create a layout form to serve as a map for implementing the model. Input variables should be placed in an input section so that these variables reflect any changes in the assumptions. Headings, labels, and cell formats should be identified at this point. Initial documentation is finalized at this step.
4. Implement the layout format using the selected spreadsheet software package. Testing and final documentation occurs at this point.

The purported use of this methodology serves as an education and training device to aid in the identification and subsequent solution of general business problems using models and presentations in an electronic spreadsheet format. The methodology presented in this paper builds upon previous work and offers an integrative set of tools that take advantage of current spreadsheet software packages. Further refinement and formal evaluations concerning the time for development, quality and utility of the applications developed using this methodology, compared to applications developed for comparable business problems using other approaches, is fertile research ground as well as significant interest to practice.

REFERENCES


