Reducing Maintenance Costs for SAS Based Systems

by Cary Toor and Wayne Beekman
Information Concepts, Inc.

Abstract

The cost of maintaining an information center oriented MIS that produces numerous user-initiated ad-hoc and standard reports can quickly outstrip the price of initial development. Maintenance of a continuously expanding and changing database compounds these problems. These costs can be substantially reduced over the life-cycle of a system by giving the responsibility for maintenance of a substantive nature -- including variable definitions, input data source formats, and output report formats -- to the users.

A MIS that does this must be designed to allow users with no expertise in system development to generate fundamental changes to the system through a simple, easy-to-learn procedure. This procedure must be built in a way that expands user confidence, knowledge, and power over system capabilities with continued use. SAS provides some of the critical tools necessary to implement such a system within reasonable time and budget constraints.

Introduction

The purpose of an information center is to allow an information seeker to directly extract and summarize data from a heterogeneous information pool. The information seeker, or end user, is already familiar with the often complex set of requirements for data manipulation. Therefore, a method of direct access to the information will allow him to extract data himself much more quickly than his requests could be fulfilled indirectly through a programmer.

In designing a Management Information System (MIS) or Decision Support System (DSS) to facilitate direct inquiries from end users, a user-oriented access method to retrieve data and perform computations must be devised. The cost of maintaining this type of system is closely related to the flexibility of the user interface. Even with an extremely flexible user interface, it is likely that any system which provides fixed ad-hoc user facilities will have a lifetime maintenance cost that greatly exceeds the initial development costs. This paper will discuss a method for developing a user-oriented system that is extremely flexible in handling ad-hoc requests from users of various skill levels as well as low in life-cycle maintenance costs.

Nature of a System: Constant Fluctuation

An effective DSS is always in the process of evolution. To be responsive to user needs, an information center must support a series of complex, constantly changing user requirements. These changing requirements can include not only additional data, but redefinition of existing data, alteration of input or output formats, and modification and addition of computational capabilities or queries. Furthermore, after these alterations are defined, they must be implemented quickly -- usually within in a few days -- to meet user demands. Initial changes are likely to generate further alterations.

For the maintenance programmer this means a constant headache. Often, he has only a few hours in which to acquire a complete understanding of the substantive nature of the changes requested. Then he must implement these alterations with little opportunity for planning or testing. This process almost guarantees that logic errors and other serious programming errors will filter into the production system. Figure 1 demonstrates the relationship between a large number of changes and maintenance costs.

To both avoid this problem and permit the end user to have immediate access to needed data, software should be designed to give the user the capability to make the changes himself. By designing the

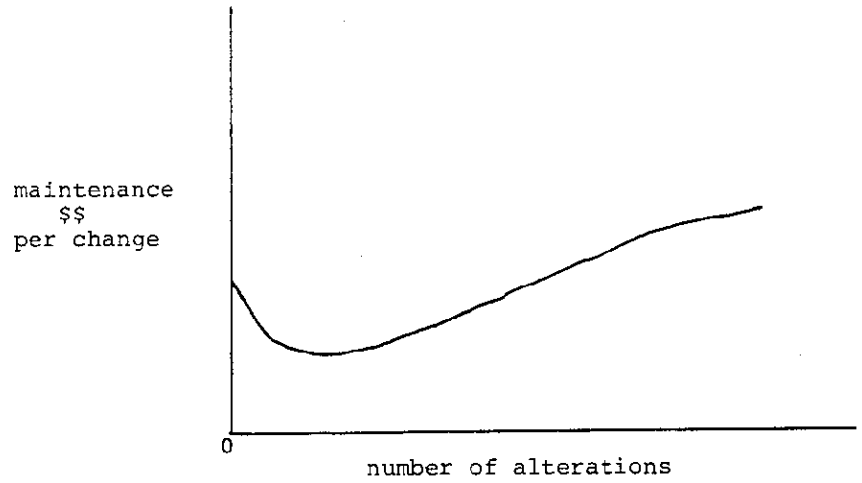

Figure 1: Cost per change in system source code.

system so that the maintenance programmer is not needed to make user-driven, substantive application changes, maintenance costs can be minimized.

Additionally, a user-driven system design can eliminate the need to change system source code, and, therefore, reduce life-cycle maintenance costs. Alteration of source code is the major factor in maintenance costs. This problem is compounded by the fact that alterations to code in one portion of a system often cause problems to occur in another.

Substantive vs Operational Maintenance

To successfully design a system with the capability described above, it is essential to clearly define system maintenance. For the purposes of this discussion there are two separate types of maintenance:

1) Substantive;

2) Operational.

Substantive maintenance is user-driven. It is the process of implementing changes that effect the capabilities of the user and the scope of the application. This includes areas such as changes in report formats, addition or redefinition of data, and modifications to retrieval or query ability.

The operational portion of the system is that which supports what the user sees. This includes applications code, screens, data access methods, and system file organization. All these elements should be transparent to the user. Operational maintenance is, therefore, heavily weighted toward getting the system to function in a manner which the user would like. Activities that fall in this maintanence category include debugging of logic errors and other coding problems, as well as reorganization of system interfaces and other changes to operational system software.

In a standard system, both substantive and operational maintenance are performed by technical support personnel. The

role of the user is limited to defining requirements to the analyst or programmer charged with implementing them. This situation engenders a number of problems. There is a good possibility that the support personnel are considerably less knowledgeable about the conceptual relations between data items than are the end users. This can cause considerable difficulty in communicating requirements in a manner that will lead to appropriate specifications being developed. In this process of communicating requirements, valuable time is lost by both the user and programmer. This time, which could be better spent by both parties in attending to duties that lie more suitably within their own provinces, plus the time and resources required to modify or develop code to meet user requirements, represents the major expense in maintaining any system.

The utility of the system can be greatly enhanced, and life-cycle maintenance costs greatly reduced, if the user has the capability to complete all substantive changes required in an simple, straightforward manner. For a system to perform these functions, it must be designed with flexibility and be user-oriented.

User-friendly vs User-Oriented

Most on-line, information center-oriented systems are designed to be user-friendly. A user-friendly system is a system which is designed for operation by an unknowledgeable user or one unfamiliar with computer concepts. A series of menus, screens, or prompts guides the user through a set of standard procedures. These procedures are hard coded and fixed within the system. A disadvantage of this design is that it greatly restricts the capabilities of fairly knowledgeable users to employ the full power at their fingertips.

Any time a fixed procedure or set of procedures is followed, making changes to the substantive portions of the procedures usually requires substantial modification of applications program code. This work consists of a combination of both substantive and operational maintenance. It will, therefore, be extremely costly and time consuming.

A system that is easy to operate and allows the user to flexibly define his own procedures and operational paths is referred to in this paper as user-oriented. This ideal system would allow the user to implement any changes or new procedures as necessary. The information seeker must have the ability to reach his goal in this flexible manner. For maximum effectiveness, the power of the complete hardware/software configuration, combined with the expertise of the user, should be constantly available. The comparative difference in maintenance costs between the traditional systems and this type of software is shown in Figure 2.

Defining an Environment

Where the typical user-friendly system guides the information seeker into a fixed process, the user-oriented system takes him into an environment. An environment is made up of a series of tools. A tool can be defined as a functional software system component that is designed to enhance the capabilities of a user in a specific logical area. Each tool has a specific purpose such as data retrieval or report generation. The tools provided are generalized to allow substantial flexibility in the user's capability.

The actual operation of a tool is transparent to the user. Access to the tool is customized for the needs of a particular user. Because an environment is designed around the needs of the user as opposed to the constraints of a system or an application, the same basic environment can be used to access and process many different types of data. A well structured environment should be totally independent of a specific application. The actual number of programs written specifically to support a single application should be minimal. Only extremely complex processes should be directly coded.

The user controls the environment and manipulates the data within its sphere not with custom written, applications specific code but with the broad tools at his command. Through a user-oriented access capability,

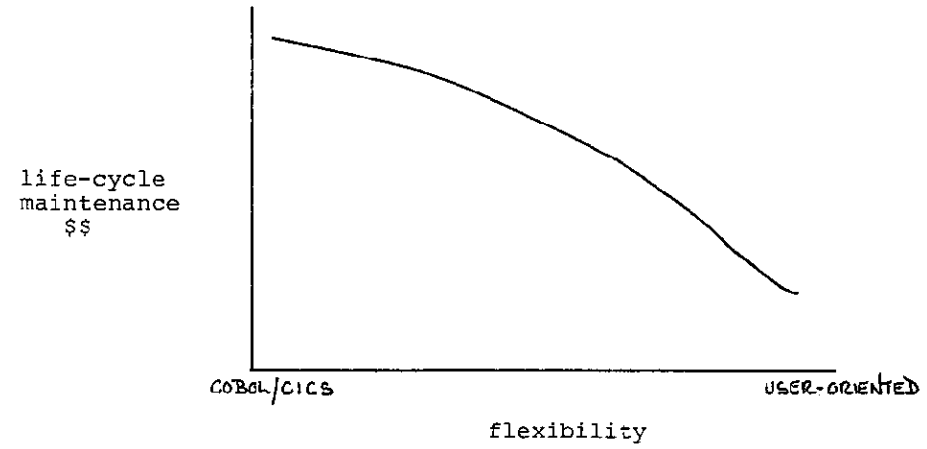

Figure 2: System flexibility vs. life-cycle maintenance costs.

customized outputs, new processes, and totally different applications can be created.

Some of the tools that will be needed by the user in this type of system include:

- An appropriate data access method;
- A query language to select subsets of the data universe;
- An analytical capability to manipulate data and create new variables from selected data;
- A generalized report formatter to create presentation quality reports; and
- A system management capability to support and control the user environment.

Most of the tools necessary to support this type of user environment are available commercially. SAS provides a substantial number of components necessary for system functionality. To build an effective user environment in a manner that is neither extremely costly, nor time consuming, it is necessary to link all functional components and tools together into a centralized operational capability.

Constructing an Environment

There are a number of components which make-up a user-oriented environment. These components are divided by logical function. Figure 3 illustrates the major components and their linkages.

It is the linkages between the functional pieces of the system which make the evironment appear as an integrated whole to the user. Because of the communications between various components, it is possible for all tools to be transparent to the end user.

As can be seen in Figure 3, the system dictionary is the heart of the environment. All components have access to and are controlled by the metadata concerning the application which is stored there. For this reason, the user-oriented

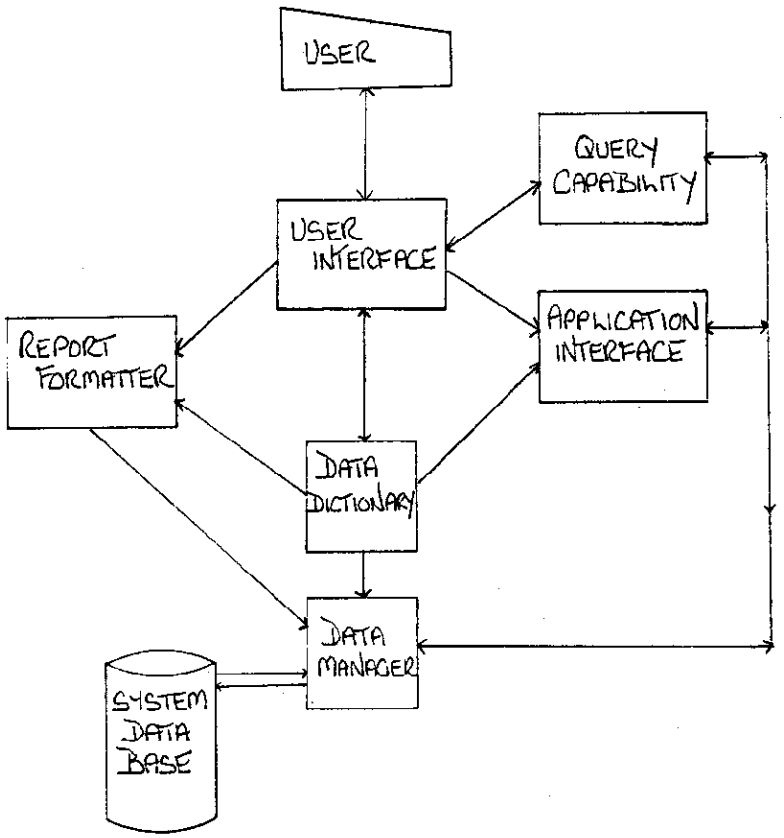

Figure 3: Major Functional Software Components of a User-Oriented Environment.

environment can be examined in two separate sections:

1) System Dictionary; and

2) Other System Components.

System Dictionary

The system dictionary is the central portion of the environment. It contains information on system files, reports, input formats, and individual data items. All data in the dictionary must be easily accessed and updated by the user through a user-oriented procedure or set of screens. It is this data which defines the remainder of the environment as well as the scope of the specific application.

The capability to access this data also constitutes the capability to greatly alter the scope of the application or increase the capabilities of the environment. The user, therefore, has a free reign to enhance or constrain the entire system through this access. Since the dictionary provides the foundation for the remainder of the system, it is essential that this capability be understood.

Applications within the environment will use the dictionary to dynamically generate the code required to perform specified user functions. To introduce a new data source into the system the user should only be required to fill out a screen-oriented form that defines the input dataset and the output data items. Information collected from this type of screen is used to update the dictionary.

The bulk of the actual application will be performed by the invocation of system tools as

their various components are necessary in conjunction with applications-specific segments of code that will be generated dynamically by the system. A program will generate this high level language -- or SAS -- application code from reading the information contained in the dictionary. There are two methods by which code can be generated.

1) Compilable code can be generated by having a program read the dictionary and write high-level language code. This code is then compiled, linked, and passed to the operating system for execution.

2) A program can be written to act as interpreter and process code line by line or plug in a set of standard parameters read from the dictionary.

When code is generated for a short application program or single use, the second method is usually more efficient. However, for resource-intensive processing, the first method is generally better. A flow diagram depicting both of these methods is shown in Figure 4.

Code generation procedures can be extremely resource intensive. This is especially true if the same code must be continually regenerated. However, the cost in manpower saved in both the initial creation of the code and the lack of need for a maintenance programmer usually makes up for this inefficiency many times over. An intelligent trade-off must continually be made between the cost of relatively inexpensive computer time and the cost of skilled programmers.

SAS is an effective language for code generation. It compiles efficiently before every execution. Its macro language allows dynamic inclusion of code created earlier in the same job allowing parameter driven execution.

Other System Components

The majority of the broad capabilities provided the user come from the outlying components of the environment. These components include:

- Data Manager;
- User Interface;
- Report Formatter; and
- Applications Interface.

The Data Manager controls all of the data held within the system. It consist of a file structure and access method which allow boolean retrievals and complete program/data independence. SAS is an acceptable Data Manager but lacks a built-in facility for indexing required for large files, does not have a simple interface to the SAS database from high-level procedural languages, and is batch oriented. A Data Manager containing these three capabilities is usually more advantageous.

The User Interface must be both easy-to-use and understand, as well as flexible. A screen formatting tool such as the IBM program product ISPF is ideal for this component. The ability to operate efficiently in an online mode as well as provide a wide range of functional capabilities is essential.

A high quality Report Formatter which produces presentation quality output is another necessary component. Automatic and controlled placement of columns, titles, and footnotes, as well as generation of summary statistics on demand is important. This is one area in which SAS has traditionally been lacking. With the introduction of the Tabulate Procedure and the enhancement of PROC PRINT in conjunction with PROC COMPUTAB, it is now possible to get by using SAS as a tool for this component.

The Applications Interface contains any code written specifically for a specific application. As an Applications Interface, SAS is an excellent tool. Its wide computational and data management capabilities make it a flexible system component ideal for the handling of modeling, simulations, and procedural processes. Because SAS code can be written quickly and is relatively simple to maintain, it is acceptable to use for hardcoding of applications specific procedures.

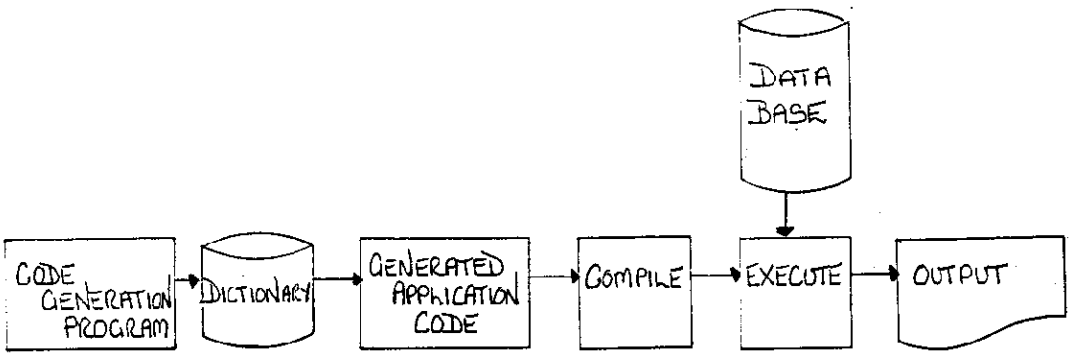

Figure 4a: Process Flow of Code Generation Procedure for Compilable Code.

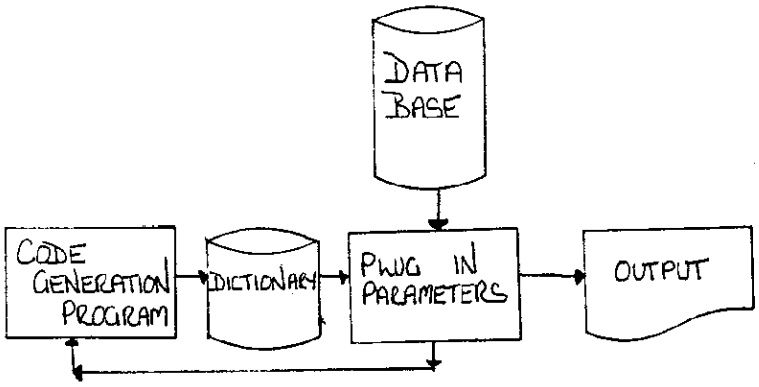

Figure 4b: Process Flow of Code Generation Procedure for Interpretative Code and Parameters.

Conclusion

By providing the knowledgeable information seeker with a flexible, user-oriented environment, it is possible to greatly increase the importance, scope, and life of a system. Not only will the development of a well structured environment be more capable of meeting user needs, but life-cycle maintenance costs will drop off sharply. The user, and not programmers with limited insight into user needs, will have the responsibility for making substantive changes to the application, thus providing him with more and better control over his information needs.

Significant decreases in life-cycle maintenance costs can be achieved through the elimination of programmers necessary to perform substantive system maintenance. By the eradication of substantive maintenance programming, it is likely that operational maintenance will also decrease. Since maintenance programming represents a major cost of any information center oriented system, the use of a user-oriented environment should significantly decrease life-cycle costs.

SAS is an ideal software tool to employ in creating the user-oriented environment. Its wide ranging capabilities as an analytic tool, data manipulation language, and data management resource make it a useful component. In addition, the capability to effectively dynamically generate code and execute it within the same job stream makes it an excellent development tool as well.

Through the use of tools such as SAS and the design of a single package of software to meet ever changing user needs, it is possible to substantially decrease the amount of programmer time required for application maintenance and, therefore, the maintenance costs over the life-cycle of a system. It is important that software be engineered to meet user, as well as machine, needs.