VIEWING SAS AS AN END-USER LANGUAGE

Steven R. Schacht, Federal Reserve Board

Abstract

The continual increase in computing efficiency has created a large demand for new application programs. The backlog of requests for programming is growing in most companies. The best way to solve this problem is for end users to write many of their own programs in an end-user language. This paper examines several properties of good end-user languages and evaluates SAS against these criteria. Several areas where SAS can be made more user friendly are noted and the way the Federal Reserve Board is addressing these difficulties is discussed.

The Problem

In many organizations, the demand for computer services is growing at an extremely rapid rate. As machine costs decline (the cost per byte is cut in half every 18 months on the average), companies buy larger and larger computers. This new capacity should permit many more applications to be automated each year. The problem is that the demand for application programming is increasing faster than data processing organizations can supply additional programmers. There just are not enough skilled programmers to go around. The backlog of applications grows larger each week.

Because of the inability to service all of the requests brought to them, data processing organizations often have to set up criteria to decide which projects will be done and which will be left unprogrammed. As a result, some computers may sit idle part of the day even though there are projects that could use this extra capacity but which are waiting for data processing to be able to get to them.

There are three possible answers to the backlog problem:

1. Increase the number of programmers.
2. Increase the output of existing programmers.
3. Provide tools so that end users can create their own applications.

The first solution appears to be an impossibility. It has been suggested that, in the absence of any increase in programmer productivity, 90 times as many programmers will be needed in ten years. There is no way this can happen.

In the area of increasing programmer productivity, many techniques have been put forth in recent years, most notably the use of structured methodology. While some improvement has been observed, the gains have not been great. Many data processing organizations have noted improvements in the 10 to 20 percent range. This alone cannot take care of the problem.

The End-User Community

The solution then must involve the expanded use of end-user languages. The word "end user" refers to the ultimate user of the computer application. At the Federal Reserve, end users include economists, program analysts, banking examiners, personnel specialists, and so on. Such people often have at best a meager understanding of the computer and know nothing of the third generation languages like COBOL, FORTRAN or PL/I. To be sure, many college graduates these days have been exposed to some sort of data processing courses, but only a few of them become proficient enough to write their own application programs.

An end user is not a single type, but may be found along a spectrum of capabilities and personalities. As an illustration of this spectrum, consider the following catalog of selected end users found at the Federal Reserve Board.

1. Numerate Norman - Economist - Studied and/or wrote FORTRAN at university. Highly numerate. Has an advanced degree or aspires to an advanced degree. Has many ideas for new computer analysis.
2. Eager Edgar - Fascinated by computers. Wants to learn everything, but has difficulty grasping concepts. Requires much help and handholding for all but most basic tasks.
3. Aspiring Ada - Research Assistant - Highly numerate. Bright, learns quickly. Views present research assistant job as a stepping stone to better things. May have learned programming in college.
4. Permanent Penelope - Research Assistant - Views work as a job, not a profession. Learns new ideas only when pressed. Can accept new methods only after "formal training."

160
5. **Occasional Oscar** — Views computer as a valued tool which is needed only occasionally. Wants to use it but has limited needs.

6. **Haughty Hilda** — Feels computers are demeaning. Prefers to have "clerk" do computations or retrievals no matter how trivial. Would be embarrassed to have a terminal in office.

7. **Timid Tommy** — Afraid of computer. Fears something might break and he will be responsible. Professional with no academic background in computers.

8. **Busy Bernie** — Analyst — Says he wants to learn about computers but never has time. When you do get him in a class, he disappears after the first session because he has to construct some tables and charts (by hand).

### The Solution

If something is to be done about the growing backlog of application programming, then most, if not all, of these user types will have to get involved in writing their own programs. Numerate Norman already does his own programming and Ada may do it as well. But the others usually use the computer either occasionally (and poorly) or not at all. In order to achieve the amount of end-user participation required, these people need software that is user friendly and nonthreatening. They must be taught how to use it in courses that are at the right level and not too technical. Then they must have assistance as they begin to develop their own programs. Consulting must be available at all times, preferably by a group of people whose sole function it is to aid and encourage users.

In short, end users must be lured into the world of computers by providing them with tools easy enough for them to use and by holding their hand as they start out on their journey.

Of course, the large, integrated applications will continue to be handled by traditional programming methods. But, it has been estimated that as many as 75 percent of applications can be handled with one of the many end-user languages.

### End-User Languages

There are many software systems on the market today that claim to be designed for use by end users. In some cases, such a claim is questionable since the software requires a greater level of sophistication than most end users possess. How do we decide if a system really qualifies as an end-user language? The first and most important requirement of a good user language is its ease of use. James Martin [1] has formulated a "two-day rule" for evaluating user-oriented languages: "We might classify something as 'suitable for end users' if typical end users can learn how to use it and obtain valuable results in a two-day training course and then not forget how to use it if they leave it for several weeks." He goes on to say that it is not necessary to master all of the system within this time period. "A desirable property of a language is that to start to use it should be easy, but that the user can continue to learn more about it and improve his skills for a long time. Languages with this property should be taught in subsets. If a language has a beginner's subset which can be well learned in one day, many users will cross the threshold from being mystified outsiders to being initiated members of the club."

A second criterion for evaluating an end-user system is its versatility. It is an axiom in the data processing world that the fewer things a system can do, the easier it is to use. If we only consider ease of use to select a system, we may find ourselves either with one that does not do all of the things we desire, or else we have to obtain several different packages. If we have many systems, we violate the ease-of-use rule since the end user will have to learn each one and probably get confused and discouraged in the process. It is better then to select a system that does many things even if it is somewhat more difficult to use. A good end-user language might include query facilities, report generation, graphics, statistics, macro facilities, time series processing, etc. Each installation must decide for itself what properties are needed.

A third standard for evaluating a package is whether the user, when creating a new applications system, can easily write a prototype of the complete system. In the automotive business, a new design would never be built without first making a prototype to ensure that the design works. It should be the same in making application programs. The user should be able to sit down at a terminal and quickly create a bare-bones version of
written specifications are usually required before he begins. In most cases, the specifications must be frozen at the start of the design phase. Often they are frozen when the user is still not completely sure as to what he wants. The more complex the application, the greater the uncertainty on the part of the user and yet the greater the need for a specification freeze. This is quite a dilemma. Then, if the coding is done in a language such as COBOL, a large number of lines of code must be generated and debugged before the user can view any results. The cost of doing this is so great that a prototype is just about as costly as a final system.

A fourth requirement of a good end-user language is the ease of changing the program. It must be possible to add a new feature to a program or to modify one part of the program without having to do anything to any of the other parts. When users begin to formulate their need for a program they usually have only a general idea of what they want. Most of the time, they decide to make changes to their original design after they have seen the first set of results. Even after the final version of the program is completed, changes will need to be done. The data items change; the law changes; the analytic needs change. If the program is written in a language that makes it difficult to modify just one part by itself, it is likely that the changes will never be completely made. Entire application systems have been abandoned because of this.

Rating SAS as an End-User Language

In most publications, the Statistical Analysis System (SAS) is listed under a heading such as "Statistical Packages." While it is true that SAS was originally designed for statistical work, many features have been added over the years which let it be viewed as a complete end-user system. While there are a few areas where SAS can be made better for the nonprogrammer, nevertheless, when it is evaluated against the four criteria discussed above (ease of use, versatility, prototyping, and independent modularity), it scores quite high.

In the earlier discussion of the ease-of-use concept, a two-day rule was suggested. The novice user should be able to produce meaningful output after two days of formal training. The "SAS Basics" course offered by the SAS Institute does just this. At the Federal Reserve Board, we offer our own in-house introductory course. It meets two hours a day for five days, a total of ten hours of class instruction. Each night's homework takes about one-half hour, making a grand total of $\frac{12}{5}$ hours. This could easily be compressed into two days. However, it is difficult to get users to agree to be away from their regular work that long. By spacing it out over five consecutive days, we get a much better response.

This introduction to SAS gives the users enough of the fundamentals so they can write programs which produce output useful in their jobs. In certain cases, users never need to learn any more than what is taught. The basics satisfy their needs. In most cases, however, they will need more. One of the things emphasized in the course is how to understand the SAS User's Guide. A user who needs more than is taught in the basic course can usually pick it up on his own from the manual. In addition, half-day sessions are offered on more advanced topics, such as SAS/GRAPH or SAS/IMS.

The second criterion is versatility. Six things were listed that are desirable in an end-user language at the Federal Reserve: query facility, report generation, graphics, statistics, macro facilities, and time series processing. We are all familiar with the capabilities of SAS in each of these areas. The weakest point is the query facility. Complicated DATA steps are often needed to select and combine data from several data bases, and many end users have difficulty with this. The R AQL facility [2] may provide an answer to the problem.

Interestingly, the report generation capabilities of SAS have become its most popular feature at the Federal Reserve. The simplicity of PROC PRINT and the versatility of the PUT statement have appealed to many users. Several veteran COBOL programmers have converted entirely to SAS for their report programs.

The third desirable feature is the ability to prototype. This feature is especially important to end users who do not have extensive backgrounds in computers. They do not recognize problems that experienced programmers would spot immediately. The end
user has to go through several versions of the program before coming up with one that works. If this trial and error approach can be done with simple prototypes composed at the terminal, the end user will get the correct version sooner and will be much happier with the process. The fact that all SAS PROCs have default parameters makes prototyping easy. Often a first attempt at a program can be constructed by writing a simple DATA step and followed by a series of PROC statements using defaults. Then parameter values can be entered after the user sees what the results will look like. Note that the use of Macros and the %INCLUDE can be helpful in prototyping. The novice user will have trouble with these, but after a couple of months should be able to handle them.

Finally, consider the modularity of SAS. Because each PROC is independent, changes in one step of a program frequently do not affect any of the other steps. Many times the only changes that have to be made to other steps is to add or delete a name from subsequent VAR statements. This property of SAS has great appeal to end users.

Improving SAS for End Users

Since SAS was not originally designed to serve as a complete end-user language for people with no previous computer experience, it has some parts that may cause problems. Some techniques which seem simple to experienced programmers confound the novice. At the Federal Reserve Board, some special features have been added to SAS which reduce the difficulties encountered by end users.

The first area that often causes problems for novices is inputting data. If all data were stored in SAS data sets, the user would never have any problem. However, in an organization like the Federal Reserve which has vast amounts of data, it is not possible to keep the data in SAS data sets. This means that users must always have a DATA step to read an OS file. But writing an INPUT statement is confusing for beginning end users since they do not understand the concept of formats.

We have been able to simplify the input process for much of the data at the Federal Reserve. We store all of the data from financial institutions in a special format called Archival Format. Each data item (variable) has a unique eight-character name associated with it. A File Description for each file in Archival Format is stored in one of the system libraries. To retrieve data from one of the archival files, the user needs only to provide the name of the file and the names of the data items he would like. A special SAS procedure, PROC ARCHIVE, has been written which reads data from archival files and creates a SAS data set. The user merely needs to enter:

```
PROC ARCHIVE IN=file OUT=esname;
ITEMS list of names;
```

The procedure loads the File Definition and reads the requested data items. This means that the user does not need to know anything about data formats. Thus, novices can begin accessing data they really need for their jobs after just a few hours of instruction in SAS.

A second area that causes problems for the end user in writing JCL statements or TSO commands. Our experience shows that the users of SAS who are not trained in other aspects of the computer seldom become at ease with JCL. Usually they learn to copy it from somewhere else and never know what it means. This is an area where we'd like to do more. Several authors [3, 4] have described SPF menus for running SAS on line that relieve the user of needing to know how to enter an ALLOCATE statement. However, most of our processing is done in the Batch mode and, hence, JCL is needed. We are attempting to design an automatic JCL generator. Until such time as we have it operational, we will have to continue providing the user with JCL to copy and a lot of help in using it.

Other things that can be done to make SAS easier for novice end users include providing easy access to help aids. It is good to see the arrival of the HELP command in SAS82. This will be a valuable tool for the end user. The excellent documentation in the various user's guides is also quite important for the continued training of the SAS user. Users should personally have a set of the SAS manuals needed for their work. In a large organization, this may seem to be expensive. But the cost of the books should be compared to the cost of additional people. Presumably, the effective usage of SAS by the end user will increase productivity and reduce the need for additional employees.

The ready access to sample SAS programs is also quite beneficial. An SPF menu can be designed which permits the user to select a sample program. The user first selects a category of programs and then,
after seeing descriptions of all of the samples in the category, selects a specific program. The SAS statements are displayed at the terminal. If the user wishes, the program is retrieved from Panvaleur and placed in the user's catalog where it can be modified to fit the user's exact needs. Such a method not only permits the new user to accomplish more sophisticated tasks, but also provides an excellent teaching tool.

Finally, an installation can make SAS easier to use by writing PROCs which perform frequently occurring tasks that would be incorporated in a DATA step by experienced programmers. This is one area where experienced data processors often lose sight of the difficulties encountered by the novice end user. People who have been writing third-generation programs for a long time are used to operating within the constraints of the language. They thrive on the challenge of figuring out how to do a new problem. They forget that it is possible to modify the constraints of SAS by creating a new PROC.

Consider the example of calculating column percentages for a set of observations. The "normal" way to do this in SAS is to use PROC MEANS to calculate the column totals, use the OUTPUT statement to put the totals in a SAS data set, merge the totals onto the detail records in a DATA step and calculate the ratios of the detail data to the totals. If we look at this from the viewpoint of the end user, it appears so complicated that many of them will prefer to do the calculations by hand. However, we can provide them with a procedure, say PROC COLPCT, that does the calculations and creates a SAS data set containing the desired percentages.

There are many other examples of situations where SAS can be modified to make it more commodious for the end user. The important point is that the people at an installation who maintain SAS need to recognize situations where new procedures should be written. Just because a task can be handled in a DATA step (or a series of them) does not mean that this is the best way to do it.

Conclusion

In the past 12 months, the Federal Reserve Board has been placing a greater emphasis on end-user application development. One part of this has been an expanded educational program in which the use of SAS is stressed. We have had excellent results from this program. Last year we had about 100 people take our introductory SAS course. Nearly half of them went on to use it on a fairly regular basis with no additional training. They report that they enjoy using SAS. Looking at the user types discussed earlier, the most enthusiastic of them are Numerate Norman and Aspiring Ada who find SAS much easier to use than the language they had been employing. The least enthusiastic are Permanent Penelope and Timid Tommy. I believe this is due to the difficulties remaining in SAS. Naturally, Haughty Hilda has not yet signed up for our course and Busy Bernie always disappears after the first session, so we do not have any results from them yet.

The most interesting results have come from the Eager Edgars and Occasional Oscars. These are users who have had some but not a great deal of previous experience with the computer and who do not know any third-generation language. We have seen many of these people become quite interested in SAS because of its ease of use and its versatility. Often, one particular aspect of SAS is just what they needed for their work, e.g., the MERGE capability or PROC COMPUTAB. Once they get interested in this one aspect, they end up learning other parts of SAS, eventually becoming quite proficient. Naturally, they need a lot of help during this process, but this is something our SAS support people are eager to provide.

Our experience at the Federal Reserve Board has indicated that many end users are extremely happy with SAS. There are ways, however, in which the SAS that comes from SAS Institute can be modified in order for it to become a more valuable end-user language.

References