COMPUTER PROGRAMMING AND AGRICULTURAL ECONOMISTS—BRIDGING THE GAP*

Marilyn G. Kletke and Darrel D. Kletke

Most ongoing research in the agricultural economics department at Oklahoma State University uses the computer at some time during its development. This is also true nationwide. It has become apparent that efforts need to be made to improve communications between economist and computer. This paper briefly presents alternatives available to the economist and advantages and disadvantages associated with each. Based on experiences of the agricultural economics department at Oklahoma State University, a method for improving communication between economist and computer programmer will be developed.

When programming needs are relatively small and uninvolved, an economist usually has three choices in his interactions with the computer. First, he has the option of using "canned" computer programs. These are routines designed and written by external agencies and distributed for general use. They are written according to a set of generalized specifications and yield a prescribed output.

The second choice is to learn a basic computer language, such as Fortran, and perform simple programming tasks himself. This allows him to satisfy his programming needs through his own abilities, thereby more exactly attaining his objectives.

The third alternative is to employ a trained computer programmer. By using this choice the economist is free to develop his economic model and the programming is left to the programmer.

There are several disadvantages associated with each of the above three alternatives. To use "canned" programs, the economist first must find a program supported by his computer installation that approximates his needs. He then must fit his problem into the somewhat rigid structure dictated by the program. Data must be formatted properly, and the researcher must worry about the program running correctly. Once run, he must be satisfied with the output, although it may not include everything the economist needs and may not be in the form he desires. Another disadvantage materializes in the tendency of some researchers to select topics to fit a specific "canned" program.

The most obvious disadvantage of the second alternative is that programming is often an uneconomical use of an economist's time. In addition, his programming abilities rarely equal those of a trained programmer. His potential for producing efficient computer programs for complex models is therefore limited.

The major disadvantage of the third alternative is the frustration an economist encounters in trying to communicate a problem to a computer programmer. This disadvantage is more serious and much harder to live with than those associated with the first two alternatives. In fact, it has driven most economists to choose one of the first two alternatives to accomplish their programming needs. As long as computer programs are relatively simple, alternatives one and two are adequate for accomplishing programming objectives. With growth in scope and complexity of many computer programs, however, the first two alternatives are no longer adequate, given the variety of tasks that are now appearing in agricultural economics.

Instructor, Computer/Analyst and Associate Professor, respectively, Department of Agricultural Economics, Oklahoma State University, Stillwater, Oklahoma.

*Journal Article Number J-3154 of the Agricultural Experiment Station, Oklahoma State University, Stillwater, Oklahoma. The authors express their appreciation to Wayne D. Purcell and Luther G. Tweeten for their constructive comments during the preparation of the paper.

As an example, suppose one wishes to program a simulation model of some production or marketing firm. This is specific enough so that a canned program capable of performing the task is not available. If the model is to include a realistic portrayal of an actual firm, the amount of data required will be such that relatively complicated data storage and retrieval schemes would have to be used. Most economist/programmers do not have time to develop programs capable of performing all the required tasks as efficiently as needed. When computer programs grow in size, efficiency becomes more important, especially if the programs are being developed for extension usage where they will be executed frequently.

Results of expanded computer needs will be increased use of alternative three: use of professional computer programmers. In order for this to be a viable alternative, frustrations and failures in economist-programmer interactions must be reduced. Thus, efforts must be made to promote effective and relatively painless communication between economists and programmers.

Little work appears to have been done as to the causes of, and solution to, friction and miscommunications that exist in this area. Nunn [4] indicates there is definitely a problem and speaks of the need to close the gap between user and programmer. Ledbetter [3] states some general circumstances leading to the breach between user and programmer. He remarks that the two groups speak different languages and have two different perspectives regarding a problem to be solved. Overton [5] avoids the problem and proposes the analyst (the economist) and programmer be one person. He then concludes that "communication problems between the analyst and the programmer are non-existent, since they are one and the same person." He does, however, admit that this has at least one serious drawback: "We should, however, consider the possibility that our programmer/analyst may not be as good a programmer as the programmer was, or . . . not . . . as good an analyst as the analyst was."

Gibson and Nolan [2] comment on the problem by saying, "the picture of EDP (Electronic Data Procession)—User relationships that emerge here is one of considerable complexity and subtlety." The situation appears to be adequately summarized by Withington [6], who notes "the development of effective, user-oriented systems depends on several very fragile processes of communication and cooperation." However, he does not go on to discuss specifics.

In short, the problem of effective communication between a user, such as an economist, and a computer programmer is recognized. However, work toward a solution to the problem is scarce and very general. The purpose of this paper is to suggest an approach which will contribute to literature and motivate additional work in this area.

THE NATURE OF THE PROBLEM

Two major reasons account for the communication gap that develops between the economist and the computer. First, the programmer is usually not an economist and, consequently, has little or no knowledge of economics. He cannot extrapolate to arrive at the bulk of economic knowledge that is carried by a word or phrase that an economist uses. Consequently, he derives much less than the full amount of information the economist intends to convey. The programmer may or may not realize that he is not understanding all he should. He has questions in his mind, but feels defensive about them and does not verbalize his problems for fear of appearing stupid, incompetent, or, at the very least, professionally naive. He decides that he can resolve on his own what he doesn't understand and proceeds to write the program with incomplete knowledge.

Conversely, it is also true that the economist is not usually a programmer. Withington states that "the user is probably not accustomed to self-analysis and rigorous documentation of his procedures" [6]. Consequently, he does not have his problem formulated in the logical detail required for programming. In addition, he has little concept of what the programming requirements for his problem are. After the programmer has begun programming, the economist usually thinks of things he needs to have included but has not yet mentioned. He brings them up in subsequent consultations with the programmer. Many times these additions represent fairly drastic program changes.

A PROPOSED SOLUTION

In order to build a communication link between economist and computer programmer, it is useful to segment the task they are attempting, and to specifically allocate each sub-task. A methodical breakdown of the task deals with the problem on two fronts: (1) it facilitates interchange of thoughts and ideas between the two, thereby eliminating associated frustration and enabling them to work together for the common goal; and (2) it helps develop a computer program which exactly meets specifications, is efficient, and which requires minimal development costs. The resulting step-wise design system may be summarized as follows:

a. Bound the problem

b. Determine the logical beginning for the problem

c. Construct a skeletal framework for the computer program

d. Outline the internal structure of the problem in a concise set of statements

e. Expand the skeletal framework of the computer program

f. Carry out the programming project.

Within each sub-task there are specific responsibilities for the economist and for the programmer. The following discussion includes both types of responsibility.

Bounding the Problem

Bounding is a key task in any programming project. Failure to do this is one of the chief sources of frustration to both economist and programmer. To accomplish this, the economist must write down in detail the exact tableau expected in his computer output. He should make up his tables complete with headings, explanatory material, and indicate where on the pages his numbers should appear. His desired output may consist of tables, summaries, lists or whatever, but a researcher must duplicate the form of the actual computer output he wishes to see. In addition, with the programmer's help, he must choose relevant computer variable names for each set of numbers included as output.

Beginning here has several advantages. When the economist designs his output in this manner, it compels him to define for himself the kinds of analyses he wants. This effectively bounds the problem and thereby delineates the scope of the computer program. This, in turn, enables the programmer to visualize entire programming results before he begins. He can choose programming techniques and begin to think about the structure of the programming task ahead. The problem will not mushroom as the project progresses since the desired output is fixed. Plans for all necessary computations will emerge in a sequential fashion from the ones just completed. There is far less chance for something to crop up at a later time that will cause major program revisions or rewrites.

Determining a Logical Beginning

The next step is to determine a logical beginning for the programming organization. The set of variables which constitutes the input serves as a foundation for the programming task and also provides a logical starting point. At this point, the economist looks at available data and determines what additional information must be obtained. He then furnishes the programmer with variable names for input data.

Constructing the Skeletal Framework

The programmer should now build a simple frame for his computer program. This consists of one or more input subroutines, several output routines and a main program that calls these subroutines. The number of output routines is dependent upon the nature of the output desired by the economist; the number of input routines is chosen by the programmer to best fit input needs. Input and output variables, whose names were chosen in earlier steps, should be placed (in the program) in one or more COMMON statements so that they may later be freely passed between routines. The programmer has assigned specific tasks to each input and output routine, and variable names have been chosen. Input and output subroutines are ready to be written. They may be assigned to a second programmer who does not need to understand the entire problem, or the primary programmer may begin the task. All variables needed in output subroutines should be set to dummy values. Output subroutines should be written to deal with them as if they had real data values. When the time comes, actual data values will be passed to the output subroutine through the COMMON statements.

Summarizing the Internal Structure

The economist and the programmer resume discussion and the former outlines the problem from input to output. Together, the two determine the basic order of processing and a general overview of steps required. More variable names will emerge in this phase, which should also be placed in COMMON statements.

Completing the Skeletal Framework

The programmer subdivides the general outline and assigns each subdivision to a subroutine. He names and structures these subroutines and briefly states what each one will accomplish. In addition, he places variable names in COMMON statements in each one, that information might flow freely. Each subroutine will initially consist of COMMON statements, a RETURN statement, and an END statement.

The Programming Enterprise

At this point, a definite computer program is emerging. The programmer has the macro view of the general structure and is ready to begin micro aspects, i.e. the subroutine-by-subroutine programming.

Programmer and economist should meet regularly for short periods of time during this phase. They should develop the content of a subroutine in detail. The programmer should then program and debug the

computer subroutine. At the next meeting, the two should discuss the current subroutine, and if ready, provide details on the next one so that the programmer can begin work on it. In this manner, the programming enterprise moves to successful completion.

As the programmer develops subroutines, he need not compile all of them each time. Once one is working properly, there is little need to spend development money to prove it again. The original dummy subroutine containing only a RETURN and an END statement can be inserted into the programming framework. Once subroutines are written and debugged individually, they can all be put into the framework. Any necessary debugging of the system can be completed. This represents a fairly inexpensive way to develop a complex computer program.

This top-down designing system provides an excellent guide for all complex programming projects. It eliminates much of the confusion in planning and aids understanding. Advantages inherent in this method are:

(1) From the beginning, there is a working computer program. Initially, it is only a main program calling dummy subroutines, but it is operational. As subroutines are developed, it slowly begins to function. It builds the output structures, even though numbers have dummy values. Gradually numbers assume true values as input subroutines, and then calculation subroutines, are added.

(2) The scope of the computer program is exactly defined in the first step when the economist designs the desired output. This enables the programmer to picture the entire program; he is able to see programming techniques he must employ as well as potential problems that he may face.

(3) Structuring a computer program in this way allows more than one programmer to work on the project. The major programmer can assign subroutines to others, who need not have knowledge of the entire project.

In this way, "chief programmer teams," as discussed by Baker [1], may be used. His basic idea is that "the chief programmer is a professional programming manager who maintains organization discipline and bears project responsibility." The chief programmer provides the interface between the economist and a team of programmers, that the work may progress more quickly.

(4) The economist does not waste his time; the programmer consults with him only until he obtains information he needs for the subroutine currently being developed.

(5) The programmer is not overwhelmed with all the technical information at one sitting; he may assimilate it in small quantities.

(6) The programmer asks questions as he comes to them in process, and small details are resolved as they arise.

(7) There is no wasted programming, since goals are precisely set prior to the programming task.

(8) This method automatically helps the economist discuss the model in a step-by-step fashion. It forces him to focus on the task at hand, thereby avoiding generalization.

(9) Structuring the program in this way makes it possible—almost easy—to make additions to the program in the future. This can be done by adding subroutines to perform the new tasks.

(10) Cost of development of the computer program is substantially reduced.

(11) Blocks (subroutines) from the computer program can be lifted out for use in other programming projects that are built in the same way.

AN ILLUSTRATION

Complex programming problems have a far greater need for this method than do fairly simple tasks. During the past year, work has been done at Oklahoma State University on a farm simulator that considers alternative disinvestment and farm property transfer strategies for estate planning purposes. Initially, the researcher and programmer met and discussed the project. They then proceeded step by step according to the method presented here.

First the economist designed his output. He needed a table depicting the beginning farm environment. This included the farm organization (sole proprietorship, corporation or partnership), assets of the farm and who owned each and operational parameters (costs, returns and physical requirements). He wanted tables for each year of the simulation that included all asset transfers (by gift, will, purchase or sell). He needed tables that would show, in the event of a death, liquidations, transfers by bequest, estate taxes, etc. Cash flow summaries by owner were desired on a yearly basis, plus associated balance sheets. Tables were needed showing projected income taxes along with itemized deductions for each owner and the farm.

The economist designed in detail all output tables and summaries he wanted. This task took considerable time and effort on his part, but it bounded the problem and made succeeding steps possible.

The economist and the programmer next chose relevant computer variable names for the numbers in the tables. The programmer chose his programming techniques and began to consider the structure of the

task. The skeletal framework that was developed yielded twenty-five subroutines. These were dummied out as indicated in step five, and the programming work was begun. After each subroutine was written and debugged, it was removed from the framework and the dummy inserted. This minimized development costs.

Some of the programming techniques that were initially chosen included the use of an asset-ownership array which was set up in a direct access file. This enabled the programmer to have access to any asset and any owner through use of a program technique using linked lists. Updates to this array were easily made when indicated by asset transfers. The environment of the farm was stored on disk at any stopping point. Thus, the simulation could be stopped and restarted from any year desired.

Implementation of this method resulted in efficient completion of the project. Development costs were substantially less than estimations made prior to the start of the project and advantages stated previously did, in fact, hold true.

There was a noticeable advantage in the attitudes of the economist and programmers involved. Each party recognized his own weaknesses and did not attempt to circumvent the other's contribution. The strengths of each were realized in the finished project.

SUMMARY

The major reason for the problems existing between economists and programmers on large programming projects may be that a method such as the one described above is not being used. Many who feel they are following a logical procedure may actually be bypassing the keys to effective communication embodied in the above method. For example, beginning with output design is critical to success of this method. This places the initial burden on the economist: he must perform the work that provides the first set of inputs to the team. It is undeniably a strong temptation for him to sketch his needs to the programmer (and ask the programmer to begin writing the computer program), saying he will return at a later time to make corrections, clarifications and output specifications. Bypassing or resequencing steps in this method allows and encourages communication problems. Each step contributes to the overall plan and cannot be skipped without potential invalidation of the method.

REFERENCES

[1] Baker, F. Terry. "Chief Programmer Teams," *Datamation*, 19:12, December 1973, pp. 58-61.

[2] Gibson, Cyrus F. and Nolan, Richard L. "Managing the Four Stages of EDP Growth," *Harvard Business Review*, 52:1, January-February 1974, pp. 76-88.

[3] Ledbetter, William N. "Closing the Gap Between DP Managers and User (non DP) Managers," *Data Management*, 12:22-3, June 1974, pp. 11-13.

[4] Nunn, John L. "Closing the Gap Between EDP and the User," *Journal of Systems Management*, 22:11, November 1971, pp. 22-25.

[5] Overton, Scott. "Programmer/Analyst: The Merger of Diverse Skills," *Personnel Journal*, 51:5, July 1972, pp. 11-13.

[6] Withington, Frederic. "Five Generations of Computers," *Harvard Business Review*, 52:4, July-August 1974, pp. 99-108.