INTRODUCTION

The programming burden has often impeded computer application, but programming time and cost have been considerably reduced by the development of advanced programming languages such as FORTRAN, COBOL, and SIMSCRIPT. The objective of the technique discussed here is to further reduce the time and effort required to produce large computer programs within specified areas.

Programming by Questionnaire, or the Program Generator technique, is a method by which numerous programs in a given area can be constructed as quickly and easily as can a few large programs. This is done by bringing together the four components that form a Program Generator:

1. A Questionnaire, written in English, defining the scope and logic of all of the programs to be generated.
2. A Statement List containing all the computer commands needed to construct any of the many programs.
3. A set of Decision Tables specifying the commands required from the Statement List as a function of the Questionnaire choices.
4. The Editor Program for processing the Questionnaire, Statement List, and Decision Tables, thus building the desired program and providing the user with a list of the data he must supply to use the program.

Of these, only the Editor is general-purpose; the other three are specific to a given area of application. An application area consists of a family of programs centered around a single basic model that has many variations. All relevant details of these variations are presented to the user in the form of a multiple-choice Questionnaire. By choosing from the options, written in English, a nonprogramming analyst can specify all the information necessary to construct his program. He then submits his choices to the Editor, along with the Statement List and the Decision Tables. The Editor constructs the program, and supplies instructions for its execution and specifications for the data required.

The Job Shop Simulation Program Generator (JSSPG) was developed to demonstrate Programming by Questionnaire. The Questionnaire for the JSSPG consists of a booklet explaining 147 options in some detail, and an answer sheet (see Figure 1). These

<table>
<thead>
<tr>
<th>CONDITION STUB</th>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Question Numbers)</td>
<td>(Answers to Questions)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTION STUB</th>
<th>ACTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Identification Numbers)</td>
<td>(Selected Statements)</td>
</tr>
</tbody>
</table>

Figure 1 – The JSSPG questionnaire answer sheet
options concern arrivals, routing and process times of jobs, decision rules for job dispatch and resource selection, shift changes, absenteeism, etc. The user specifies which options apply to the model he wants to generate. His choices, when given to the Editor along with the Statement List and Decision Tables, are translated into a SIMSCRIPT computer program.

Many different simulations can be generated in this manner. While there are not a full $2^{17}$ different models, since not every possible combination of answers is permissible, at least $2^{20}$ models are possible — more than one billion. The work required to develop the JSSPG, however, was comparable to the effort required to build a few large models.

**General concepts**

**The questionnaire**

The Questionnaire is the only part of a Program Generator the user sees. He does not need to know anything about the other components, but only to understand what he wishes to model, answer the Questionnaire, and supply input data. One important characteristic of the Generator concept is that the user must specify only the basic structure of his model and a few numerical data on the Questionnaire; he supplies the rest of the data when the Generated Program is executed. The Editor specifies the type and form of these data since they will vary from program to program, depending upon the options chosen on the Questionnaire. The only numbers required on the Questionnaire are those needed for storage allocation.

**The statement list**

The Statement List consists of all the commands, partial commands, groups of commands, and special instructions required to build any program that may be described on the Questionnaire. Since the Editor produces SIMSCRIPT programs, the Statement List also includes all the information peculiar to SIMSCRIPT, such as definitions of variables and initialization data, and special information required in the Generated Program.

The commands in the Statement List could have been written in any computer language, from machine language to the higher languages such as COBOL and SIMSCRIPT. Since the effort involved in building a Program Generator depends largely on the length and complexity of the Statement List, a Generator is easier to build using a higher language because of its simplicity, flexibility, and ability to accomplish a task with fewer commands. If any language other than SIMSCRIPT were used, the Editor would require modifications whose extent would depend upon the complexity of the language and its similarity to SIMSCRIPT.

The commands look exactly as they would in any other program, except for their Identification Numbers. If a command has no Identification Number, it is considered part of the preceding command that does have a Number. Thus, whole groups of commands can be called from the Statement List by simply selecting one Identification Number.

A Statement List command may also differ from a normal command in that it may be only part of a complete command. For example, in constructing the Statement List for the JSSPG, it was found that a given phrase (such as FIND FIRST) could be controlled by a variety of control phrases (such as FOR EACH JOB), depending upon which options the user chose on the Questionnaire. Rather than repeat the phrase common to all options, it is convenient to give it an Identification Number and number each of its modifiers. The Editor then combines all parts into a single command.

In addition to commands, the Statement List contains other information. For the SIMSCRIPT language, this includes the definition cards that specify the program variables and the initialization cards that specify the array sizes in memory. Also, the user is told what data he must supply in order to use the Generated Program, and the computer is supplied with control cards for execution. All four of these components (the definition and initialization cards, the input data requirements and instructions for its format, and the control cards) are cards in the Statement List, just like the commands, and are selected, based on the user's answers to the Questionnaire, in the same manner as the commands.

**Decision tables**

The Editor employs the Decision Tables to decide, based on the responses to the Questionnaire, which cards in the Statement List to include in the Generated Program. In building the Statement List, the programmer must have in mind the various combinations of commands needed to satisfy all the options that the user may select on the Questionnaire. The Decision Tables are merely a formal statement of the relationship between the Questionnaire choices and the programming statements. They give the Editor a set of rules, in a standardized manner, so that it can choose the right statements and handle them properly.

Decision Tables are divided into four quadrants, usually named as in Figure 2. The entries in the Condition Stub are keyed to questions on the Questionnaire; those on the Action Stub correspond to Identification Numbers in the Statement List. Thus, the Decision Tables link the Questionnaire and the Statement List.
The Conditions indicate the responses to the Questionnaire that are required for action to be taken. The Actions indicate which statements (whose Identification Numbers appear in the Action Stub) should be included in the Generated Program, given that certain questions or combinations of questions have been answered in a certain way.

Figure 3 shows a sample Decision Table from the JSSPG. Each column in the Conditions field represents a single combination of feasible responses to the questions listed on the Condition Stub. For example, there are four allowable combinations of responses to the five questions shown in Figure 3, as the nature of the questions precludes any other combinations. The first column of two N's would apply if neither D 9 nor D 10 on the Questionnaire was chosen, i.e., the user had, in effect, answered them both ("No."). The second column indicates a "Yes" response to B 3 and apparent indifference to the other four questions. In actuality, the second column would apply only if either D 9 or D 10 had been answered "Yes." The column order is significant in that the first column found to match the answers on the Questionnaire, when scanning from left to right, will be the column that applies. Thus, the second column appears to imply indifference to D 9 and D 10, but it would only be chosen if both B 3 and either D 9 or D 10 were answered "Yes." Likewise the third column could be chosen only if either D 9 or D 10 was "Yes," B 3 was "No," and B 12 was "Yes." A similar statement could be made for the fourth column.

When a column matches the answers on the Questionnaire, the actions to be taken are found in the same column in the Action portion of the Table, indicated there by X's. For example, if neither D 9 nor D 10 was selected, the Editor would include the statements with Identification Numbers 125 and 781 in the Generated Program. The minus in front of statement 781 tells the Editor that card 781 is not a regular command, but rather is special information (in this case, a special definition card).

The fifth column in the Conditions can be reached only if the Questionnaire answers are inconsistent; i.e., no feasible combination of answers to the questions has been realized, implying that the user has
**Figure 3 – Sample JSSPG decision table**
made an error in completing the Questionnaire. The Editor, upon finding no Actions to be taken, rechecks the Conditions. If the Table indicates indifference to all questions, the Editor writes an error message, telling the user his responses are inconsistent. It also gives him a list of the questions involved and his responses to them. It is, however, permissible for there to be no Actions, or for there to be a final blank column reached in the Conditions by a process of elimination, thus implying apparent indifference to all questions, but not both.

The Editor

In the broadest sense, the Editor is the computer program that translates the user's responses to the Questionnaire into a computer program. The Editor is written in SIMSCRIPT and is capable of producing only SIMSCRIPT programs of either a simulation or nonsimulation nature.

Operation

The Editor treats as input the other three parts of the Program Generator: the Questionnaire, the Statement List, and the Decision Tables. While their contents will differ with Program Generators written for different application areas, their logic and construction will be the same. Thus, the current version of the Editor can be used for other Program Generators in any area of application, if the language used for the Statement List is SIMSCRIPT, and if the principal components are constructed as outlined here.

The Editor has four functions:

1. To translate the answers to the Questionnaire into a computer program.
2. To check the answers to the Questionnaire for completeness and consistency.
3. To supply all control cards and dictionary information necessary to execute the Generated Program.
4. To provide a list of the names, meanings, and dimensions of the variables whose values the user must supply to the Generated Program during execution.

Besides a printed listing of the program, the Editor can supply a corresponding deck of cards that contains all the required control cards if the Program Generator is in the operational phase and not the development stage. The user need only place the required input data at the back of this deck where indicated and submit it to the computer for compilation and execution.

Other techniques

In addition to the obvious savings of time and effort, at least two other important benefits accrue from the automatic preparation of computer programs. Since available features are frequently presented in tabular form, they can serve as a checklist to remind the analyst of facets of his problem that he may have overlooked. Also, with a new or modified program readily obtainable at small cost, the analyst will frequently investigate less obvious alternative solutions that might not be considered if an existing program had to be modified. The desirability of these benefits has prompted many attempts at automatic program preparation.

A number of different techniques have been developed. In one, called the “Modular” approach, the user builds a program by selecting and linking a number of pre-programmed subroutines. This approach is often unworkable; it is difficult to make the subroutines logically compatible and the method necessitates a large library of subroutines. The approach may prove feasible if the set of options presented to a user is relatively small, such as in a sort program generator.* Because the Program Generator uses decision tables and compiles programs from individual commands rather than from larger modules, it alleviates most of these difficulties, allowing the user a much wider range of options.

The “Generalized Model” approach uses one large program containing all possible options; those to be executed in a given run are specified by parameter assignment in the input data.† The principal difficulty with this method is the inefficient use of computer time and memory space. The program must continually interrogate the parameters the user specified, using the results to decide which options are to be performed. If the options are very numerous, this process takes up a significant portion of the total running time. The Program Generator escapes this difficulty; it checks the options only once, at the time the program is generated.

The Generalized Model uses computer memory space inefficiently because the memory must contain all the options and the logic for choosing among them during the program's execution. Memory size therefore limits the number of options available. Since a Program Generator constructs only the code necessary to execute the chosen options, the generality

*See, for example, IBM 7090 Generalized Sorting System, IBM Systems Library, File No. 7090-33.
†One illustration is: The Job Shop Simulator, Mathematics and Applications Department, Data Systems Division, IBM.
of the approach is usually not limited by available memory space.

Program generators are not new. An example of a previous generator is the GQS.† As a consequence of how the user specifies his desired model and of the method used to generate the program, however, the range of options that can be offered in any one such compiler is very limited, as compared to the Questionnaire method.


In summary, the following features of Programming by Questionnaire distinguish it from other methods of program generation:

1. An English language questionnaire, requiring no special knowledge of either detailed programming languages or format specifications.
2. Generation of computer programs that are as efficient as possible (given the limitations of both the language used for the Statement List and the person building the generator) in terms of computer memory space and computer time.
3. The ability to provide an almost unlimited range of options in any one generator.