Discussion and correspondence
Q charts—a method of specification*

G. Duncan

Rand Information Systems Ltd, 1 Parkshot, Richmond, Surrey TW9 2RD

The purpose of this note is to outline an approach to software specification and documentation which the author uses for both system and program design. It is not proposed that the ensuing technique be applied in all circumstances; alternative presentations such as flowcharts may be more suitable for particular problems.

(Received March 1979)

The term Q Chart, standing for Question Chart, was derived from the emphasis given in the method to the questions or conditional branches of the procedure being specified. These questions or conditional branches control the activities actually performed and can be regarded as the real skeleton of the procedure. The non-question processing steps can be viewed as being hung upon the question skeleton. This view of a procedure as a framework of questions is borne out by the errors or bugs which are usually found in complex programs. Frequently, the processing step itself is not incorrect but rather the error tends to lie more in the fact that the step is being performed at all, indicating that either a question has been omitted or a conditional branch has been wrongly designed. This does not mean that the processing steps are unimportant, but rather that the complexity of a procedure is primarily dependent upon the network of the questions and not upon the nature of the processing steps performed.

Unlike conventional flowcharts, a Q Chart is written rather than drawn. Although this distinction may appear to be trivial, it does highlight a fundamental difference in approach. A Q Chart is composed of sentences or phrases written in conversational language; it does not require templates; it may be handwritten or typed and can be used at any level of specification. Through the use of embedded comments even a detailed program code specification can be made self documenting.

Basically a Q Chart consists of a set of consecutively numbered sections. Each section is composed of statements which may be processing steps, comments, jumps or questions. Unless the flow is diverted by means of a jump or question, the implied flow through the chart is always to the next statement, be it in the same section or in the next section. There is no definite limit to the size of a section, but the ease of reading, comprehension and maintenance can be impaired if a section extends over more than two pages. The first statement of each section is given a subnumber of zero, i.e. 1.0, 2.0, etc. The other statements are not numbered unless an explicit jump is made to a statement other than the first in a section, in which case that statement is given a unique subnumber in the left margin and a series of dashes is drawn connecting the label and the statement. The jump format consists merely of 'JUMP section.subnumber'. It is a principal design objective of Q Charts that although jumps may be made to any subnumbered statement within the same section, jumps to other sections should only be made to the first statement of the other sections. If this rule is adhered to, then a programmer at either the design or maintenance level is assured that a section can be redesigned without affecting any jumps which may have been made from other sections.

The structure of a Q Chart permits comments to be inserted at any point in the system without breaking the natural flow of the process. Comments consist of any specific sentences, phrases, numbers, program variable references or explicit expressions which may improve the understanding of the process being specified. They are enclosed within parentheses and can appear anywhere within a statement. The author has found it useful when writing detailed program level Q Charts to comment virtually every line. The commented Q Chart is the main program documentation with cross referencing comments in the program identifying the start of every section.

The physical layout of the Q Chart is important. The numbered sections of the chart are organised as block structures. All statements at the same block level have the same indentation and are preceded by a dash. This also applies to a question and the identification of the answers associated with the question. However, the statements which form the various answers are treated as being at a lower block level and have a greater indentation, being aligned to the right of the answer identifier. The answer identifier is placed within parentheses and may be labelled with a subnumber if the action at that answer is the target of a jump. In order to improve readability, a vertical line in the form of an elongated square bracket is drawn to the left of a question and its answer identifiers. Apart from merely linking the subblocks associated with the question, the vertical line is a valuable aid to the location of the relevant reply, especially in cases where the answers contain further questions and answers.

There are three types of question which can be used within a Q Chart structure, namely dichotomous yes/no questions, comparative questions and multiple reply questions. By convention, the yes reply for dichotomous questions always appears first with the yes reply flow jumping around the no section. This type of question translates readily into the construct IF . . . THEN . . . ELSE . . . . The comparative question is framed in the form ' – compare A with B (commented)'. The order of the three answers is always < (A less than B), = (A equals B) and > (A greater than B). Unless a specific jump is included to force continuation, the flow is assumed to jump around the intervening answers. The format of the multiple reply question depends greatly upon the nature of the question being asked.

Since it is frequently necessary in both detailed and high level specification documents to indicate that a looping action is to be performed over a set of statements, a specific Q Chart format has been designated for loops. The control statement at the start of the loop is specified as:

section.subnumber – FOR action; AT END label

The subnumber should not be .0 since confusion can arise between looping and jumping to the start of the loop. The 'action' can be any meaningful phrase such as 'all columns' or 'J starting at 10 and decrementing by 1 until 0'. The 'label' is the identifier of the next statement to be done at the conclusion of

*This technique was originally developed when the author was with the OR Unit of ICL Dataskil, Reading, England.

the loop. In the statements which follow, the looping activity is invoked by the phrase ' – LOOP TO section.subnumber' where section.subnumber is the identifier of the FOR statement. Assuming that it is within one section, the whole of the loop is connected by a square bracket line.

Frequently, the end of a page is reached at an awkward point in a section with perhaps three or four questions still open. The linking lines are continued to the bottom of the page and labelled in alphabetical order. The required number of lines is drawn at the top of the ensuing page and labelled in the same order. The chart is then continued without further modification.

Example

1.0 – how much money has been raised?
- – <£10,000) (get more volunteers) JUMP 5.0
- – (£10,000 – £30,000) (continue fund raising) JUMP 4.0
- – (£30,000 – £50,000) (cut back on campaign) JUMP 3.0
- – (> £50,000) (consider building)

2.0 – (consider building)
- – compare money collected with builder's contract price?
- – (<) – (insufficient money)
  - – have all of the pledges been collected?
  - – (yes) (continue fund raising) JUMP 4.0
  - 2.2 – – – – (no)
    - FOR all pledges not collected; AT END 2.0
    - – JUMP 9.0 (INSERT SECTION 9 AT THIS POINT – AMENDMENT)
    - – LOOP 2.2
- 2.1 – (=) – (sufficient funds – collected or pledged)
  - – sign the fixed price contract
  - – JUMP 7.0 (have a celebration drink)
- – (>) – (excess funds)
  - – consider improving the quality of the inside furnishings
  - – JUMP 2.1

To the Editor
The Computer Journal

Sir

Vector approximation to curves

D. H. McLain's 'Vector Approximation to Curves', Algorithm No. 100 (*The Computer Journal*, Vol. 21 No. 2) should be required reading for all engaging in computer graphics. Although the principle is not new (I have used it myself since 1969) McLain's solution is particularly neat and an object lesson in the avoidance of trig functions. Even so, it could be improved in two respects:

(a) The author acknowledges that a substantial part of processing time is spent in the SQRT routine, yet for every point in straight and near-straight lines he calculates the square root twice—first to update the upper boundary and then the lower.

(b) As it stands Algorithm 100 is unsuitable for some applications (e.g. my own, which is cartography) because a long narrow closed figure, having width equal to or less than the permitted deviation, would collapse into a point. To prevent the disappearance of ridge contours and the truncation of spurs, I make an additional test and terminate the vector if it begins to shorten.

A similar but 3-dimensional vector approximation is used when reducing photogrammetric data captured from stereoscopic models. Here the 2-dimensional version would of course invalidate subsequent operations such as spatial transformation or the extraction of vertical sections.

Yours faithfully,

K. M. KEIR

Hunting Surveys Limited
Elstree Way
Borehamwood, Herts WD6 1SB
8 June 1978

To The Editor
The Computer Journal

Sir

Decision tables

An important topic in the paper on decision tables by R. Maes (1978), is the conversion of an arbitrary flowchart to a program-oriented decision table. A paper which I wrote in conjunction with K. Hutchings (Dwyer and Hutchings, 1977) was known to Maes, but the apparently inevitable delays of publication have prevented him from citing it. I therefore feel justified in bringing the paper to the attention of your readers.

The paper describes a COBOL-based decision table language and processor called COPE. It details a number of innovations. One of these is a direct method of converting flowcharts to program tables.

The approach is as follows: If a flowchart is loop-free, it has a finite number of paths from its entry point to its exits point. The content of each condition box and action box is coded as a stub. A list of stubs is made up, in the logical order in which they appear in the flowchart. Each path through the flowchart is described by checking off those boxes which are on it, thus building a table. (There is no requirement for all conditions to precede all actions.)

Whereas a simple 'X' serves to indicate that an action appears on a path, a condition box is more complex. Because a decision box has

```
*MAES-FIG-1. NOTE TABLE.
*
*    1 1 1 2 2 2 2 2 2 2 2 3 3   GROUP.
*    1 2 3 - - - - - - - - - -   IS I1 .. IN.   = < >.
*    - - - 1 1 1 2 3 3 3 3 - -   IS I1 .. I2.   > = <.
*    - - - 1 2 3 - - - - - - -   IS I2 .. UGR.  > < =.
*    - - - - - - - 1 1 2 3 - -   IS I2 .. OGR.  = < >.
*    - - - - - - - N Y - - - -   IS A(I2) = A(IN).
*    - - - - - - - - - - N Y     IS A(I1) = A(IN).
*    - - X - - - - - - - - -     PERFORM A.
*    - X X - - - - - - - - -     PERFORM B.
*    - - - - - - - - - X - -     PERFORM C.
*    - - - - - - - - - - X - -   PERFORM D.
*    X - - - - X X X - - - - -   PERFORM F.
*    - - - X - - - - - - - - -   PERFORM G.
*    - - - - X - - - - - - - -   PERFORM H.
*    - - - - X - - - - - - - -   PERFORM J.
*    - - - X X - - - - X - - -   PERFORM E.
*    - 2 2 3 3 - - - - 3 2 2 -   NEXT GROUP.
*    NOTE END OF TABLE, 17 CORRECTLY CODED ROWS, 13 RULES.
*    ENTRY IN ROW  2, RULE  3 IS A COMMENT - NO VALID ALTERNATIVE.
*    ENTRY IN ROW  4, RULE  6 IS A COMMENT - NO VALID ALTERNATIVE.
*    ENTRY IN ROW  3, RULE  9 IS A COMMENT - NO VALID ALTERNATIVE.
*    ENTRY IN ROW  5, RULE 11 IS A COMMENT - NO VALID ALTERNATIVE.
*
 MAES-FIG-1.
 MAES-FIG-1-1.
     IF I1 = IN; GO TO MAES-FIG-1-2430.
     IF I1 < IN; GO TO MAES-FIG-1-2131.
     PERFORM A.
 MAES-FIG-1-2131.
     PERFORM B.
 MAES-FIG-1-2.
     IF I1 > I2; GO TO MAES-FIG-1-1633.
     IF I1 = I2; GO TO MAES-FIG-1-2430.
     IF I2 = OGR; GO TO MAES-FIG-1-1837.
     IF I2 < OGR; GO TO MAES-FIG-1-2833.
     PERFORM C.
     PERFORM D. GO TO MAES-FIG-1-2.
 MAES-FIG-1-1837.
     IF A(I2) = A(IN); GO TO MAES-FIG-1-EXIT ELSE
         GO TO MAES-FIG-1-2430.
 MAES-FIG-1-1633.
     IF I2 > UGR; GO TO MAES-FIG-1-2533.
     IF I2 < UGR; NEXT SENTENCE ELSE GO TO MAES-FIG-1-2430.
     PERFORM H.
     PERFORM J. GO TO MAES-FIG-1-2833.
 MAES-FIG-1-2533.
     PERFORM G.
 MAES-FIG-1-2833.
     PERFORM E.
 MAES-FIG-1-3.
     IF A(I1) = A(IN); GO TO MAES-FIG-1-EXIT ELSE
         GO TO MAES-FIG-1-2.
 MAES-FIG-1-2430.
     PERFORM F.
 MAES-FIG-1-EXIT.
     EXIT.
*
```

Fig. 1

```
*MAES-FIG-1. NOTE TABLE.
*
*    1 1 1 2 2 2 2 2 2 2 2 2 3 3   GROUP.
*    1 2 3 - - - - - - - - - - -   IS I1 .. IN.    = < >.
*    - - - 1 1 1 2 3 3 3 3 - -     IS I1 .. I2.    > = <.
*    - - - 1 2 3 - - - - - - -     IS I2 .. UGR.   > < =.
*    - - - - - - - 1 1 2 3 - -     IS I2 .. OGR.   = < >.
*    - - - - - - - N Y - - - -     IS A(I2) = A(IN).
*    - - - - - - - - - - - N Y     IS A(I1) = A(IN).
*    - - X - - - - - - - - - -     PERFORM A.
*    - X X - - - - - - - - - -     PERFORM B.
*    - - - - - - - - - - X - -     PERFORM C.
*    - - - - - - - - - - X - -     PERFORM D.
*    X - - - - X X X - - - - -     PERFORM F.
*    - - - X - - - - - - - - -     PERFORM G.
*    - - - - X - - - - - - - -     PERFORM H.
*    - - - - X - - - - - - - -     PERFORM J.
*    - - - X X - - - - X - - -     PERFORM E.
*    - 2 2 3 3 - - - - 3 2 2 -     NEXT GROUP.
```

Fig. 2

more than one exit, the table must stipulate which exit is being followed. This is done by writing 'Y' or 'N' for two-way decisions. The more general case of multi-way decisions is done by numbering the exit paths. Such a 'multi-choice' condition is expressed by a generalised stub, followed by a list of parameters which apply to the specifically numbered exits. (This notation is a useful compromise between limited—entry tables, which tend to be cumbersome and extended—entry tables, which tend to fit too few rules on to a coding sheet.)

The technique of describing a flowchart by its paths breaks down when there are loops. But such flowcharts can always be described by dissecting them into components which are loop-free. This is done by assigning lables to selected points in the flowchart. There must be a label at the entry point of the flowchart, and at least one label in each loop. Beyond this, the choice is arbitrary.

The flowchart can now be documented by tabulating the flowpaths as follows:

1. The starting label (referred to as the GROUP)
2. The list of conditions and actions on the path
3. The terminating label (referred to as the NEXT GROUP).

To apply this technique to Maes' Fig. 1 four points can be labelled. The entry point can be labelled '1'; the conditions I1::I2 labelled '2'; and A(I4)::A(IN), '3'. By convention the exit point is always labelled '–'. The result of this labelling is shown in **Table 1.**

There is an interesting analogy with the standard technique used to convert an arbitrary program into a 'structured' program. This technique introduces an auxiliary variable to enable and program to be reduced to a single loop containing a single case statement.

We can, if we wish, imagine the GROUP to be just such an auxiliary variable. The first action in each rule tests it, and the last action either exits the program, or sets a new value in the variable and continues the loop. The conversion of a flowchart to a COPE decision table, and its 'structuring' with an auxiliary variable are closely analogous. (COPE does not actually use auxiliary variables. It will produce a COBOL program with the same flowchart use auxiliary variables. It will produce a COBOL program with the same flowchart as the original problem. The method is explained in Dwyer and Hutchings (1977.)

Because there is freedom in choosing the points to be labelled, different tables can be derived from a given flowchart. The various techniques reviewed by Maes correspond to COPE tables with different labellings. Thus Maes' Fig. 5 is the case of labelling all conditions. Table 5 is similar, except that the single auxiliary variable representing the label has been replaced by a set of two-way switches. Table 6 corresponds to labelling the entry point of each loop.

COPE can therefore be used in ways analogous to each of these examples. This flexibility may be exploited to choose a labelling which best seems to document the problem.

Contrary to Maes' general conclusions, COPE has proved to be an excellent COBOL programming tool (Dwyer, 1978). It is claimed to save about 50% of PROCEDURE DIVISION coding.

COPE is written in ANS COBOL. A source tape is available from the writer for a moderate charge. The program can be expected to work on most COBOL systems.

Yours faithfully,

BARRY DWYER
School of Mathematics and Computer Science
South Australian Institute of Technology
North Terrace
Adelaide, South Australia
2 February 1979

References

DWYER, B. and HUTCHINGS, K. (1977). Flowchart optimisation in COPE, a multi-choice decision table, *Australian Computer Journal*, Vol. 9 No. 3, p. 92.

DWYER, B. (1978). Experience with COPE, a Multi-choice decision table processor, *Proceedings of 8th Australian Computer Society Conference*, Vol. 1, p. 302.

MAES, R. (1978). On the representation of program structures by decision tables: a critical assessment, *The Computer Journal*, Vol. 21 No. 4, p. 290.

To The Editor
The Computer Journal

Sir

Pseudo-random sequences of Mersenne prime residues

The paper by G. W. Hill (*The Computer Journal*, Vol. 22, pp. 80-85) includes description of a FORTRAN function RESIDU(K,KN) 'which replaces the value of KN by K*KN(mod M), leaving K unchanged'. The author then, however, uses X=RESIDU(I(J), I(J)).

The 1966 Fortran Standard (clause 8.3.2) says 'If a function reference causes a dummy argument in the referenced function to become associated with another dummy argument in the same function . . . a definition of either within the function is prohibited'.

The 1978 Fortran Standard repeats this restriction, in slightly modified words, in clause 15.9.3.6.

Yours faithfully,

I. D. HILL
MRC Clinical Research Centre
Watford Road
Harrow, Middlesex HA1 3UJ

Book review

Computers and Commonsense (2nd Edition) by R. Hunt and J. Shelley, 1979; 166 pages. (*Prentice-Hall*, £3·50)

This new edition of a book first published four years ago has been updated by new technology and by the mention of new applications and social issues. As computers influence the life of the ordinary man more and more it becomes much more important to deal with these issues. Five pages may be better than nothing but it is very little compared to the space allocated to the arithmetic and logical unit with which few users of computers are directly concerned. The social issues at work, where top management still has a tendency to lay down the 'requirements' of a new system without consulting those who actively operate the former clerical system, has only been touched upon. The position of the Trade Unions and the welfare of the individual worker as a human being are not really investigated. Is compulsory idleness for the unskilled going to be an inescapable future? Trying to train them to higher levels of skill is an idealistic aim, especially if it involves a loss of personal identification.

This book is well put together, with its cross referencing and its updated bibliography, and such criticism only points towards areas where commonsense is still to be developed. The need for a new edition within four years shows the demand for this paperback which provides nine pages of introduction to BASIC in addition to the wide picture required by everyone who wants an introduction to computers or wishes to develop a sense of proportion in these confusing developments.

PHILIP GILES (Stirling)