Correspondence

Dear Sir,

Human Performance in Interactive Graphics Operations
[**R. A. Reynolds**, *The Computer Journal* **26**, 93 (1983)]

I find Reynolds' note interesting but consider that there are two areas which are unfortunately not covered.

(a) It is not clear whether the results quoted show a statistically significant difference between sessions of up to one hour and those over one hour. A minor further point here is whether the data should have been grouped in whole hours or whether a finer analysis of length of session would have shown, for example, that very short sessions were atypical.
(b) As to the interpretation of the results the author mentions that some operator errors will be accepted as valid commands but suggests that the number of decisions taken in any session should be a good measure of operator performance. From personal experience in interactive non-graphics work, the number of mistakes increases after a certain length of session; this would tend to reduce the difference between less than one hour and longer sessions. In any case, it is presumably appropriate decisions that constitute a good measure of operator performance rather than total decisions; one would want to distinguish between the operators who made the same total number of decisions but significantly different numbers of good decisions.

Yours faithfully
G. PHILLPOTTS
Statistical Department
Home Office
Queen Anne's Gate
London SW1H 9AT
UK

R. A. Reynolds replies
I cannot agree that it is useful to distinguish between individual operator performances. The intention of the exercise was to establish the typical working pattern of an average operator in order to optimize rostering.

I must of course agree that mistaken decisions should not ideally be included in a measure of performance. However, in a modern interactive draughting system such as the one described there are normally several ways of carrying out the same operation, some of which are more efficient than others in certain circumstances. The process of detecting mistaken decisions is therefore not as straightforward as is implied. In an observation of a real-life situation on the scale of the one described, where over 140 000 commands were issued, it is probably impracticable.

A laboratory experiment in which a number of operators carried out similar tasks could of course be devised, but great care would have to be taken that these tasks did in fact reflect a realistic working pattern.

Further comments from G. Phillpotts
I have the following further observations, in the order in which Mr. Reynolds has replied.

(a) I suppose my original letter may not have been sufficiently clear as regards distinguishing between operators who had different productivity in terms of good decisions (last sentence of my letter). My point was that just as one would in day-to-day management wish to distinguish between operators on this basis, so should statistical measures of the aggregate performance of a number of operators try to take this into account.
(b) I agree this is a problem.
(c) It occurs to me that an experiment along the following lines might meet the combined requirements of realism and good measurement:

Divide the operators into experimental groups. Within each group, they would work the same working pattern with a fixed length of terminal session. The different groups would be assigned different session lengths. Incoming work would be randomly allocated to groups and the outcome would be the time taken to 'deal' with a piece of work.

Dear Sir,

The Tower of Hanoi as a Trivial Problem

I must take issue with M. C. Er when he calls the Tower of Hanoi problem 'intractable'.[1] When the problem is used in introductory texts on computer science, any difficulty which occurs is more likely to be because the trivial nature of the problem leads to a lack of appreciation of the power of recursion, rather than because of a lack of insight. At least recursion emphasizes the structure of the problem.

More importantly, although Er provides a non-recursive analysis of the problem, it is insufficient. He analyses the *moves* of the discs and not the *status* of the discs after each move.

If we tabulate for each disc which peg it resides on after each move the solution is obvious. Using zero based disc and peg numbering the peg on which disc j resides after move x is given by

$$P_j \equiv (-1)^j \left\lfloor \frac{x + 2^j}{2^{j+1}} \right\rfloor \pmod 3$$

where $\lfloor\ \rfloor$ is the floor function. All the properties of the Tower of Hanoi problem follow trivially.

Yours faithfully
R. J. HEARD
Programmer
Queensland Institute of Technology
G.P.O. Box 2434
Brisbane
Queensland
Australia, 4001

Reference

1. M. C. Er, A representation approach to the Tower of Hanoi problem. *The Computer Journal* **25**, 442 (1982).

Dear Sir,

System Prototyping

The idea of system prototyping proposed by Dearnley and Mayhew (*The Computer Journal*, **26**, 36 (1983)) seems to me very powerful. As the authors say, prototyping is widely used for industry systems, but not much for computing systems, yet a computer with modern software tools is ideal for modelling real-world situations, including other computing systems, and is re-usable. Further, the task of system modelling is much more congenial than writing paper specifications, which leads to higher productivity.

I would like to add support to system prototyping by describing two cases in which I have been involved, both of which worked very well.

In the first, some years ago, a model of a process involving parcels of random dimensions being packed into rectangular containers, clearly demonstrated the superiority of certain kinds of strategic packing over random placement, with very little software design effort and small computer running costs.

In the second case, more recently, a model of a system involving the reservation by computer users of terminals, communications channels, and interactive processes on several service computers was created using Fortran and IDMS on an ICL 2970 mainframe. The intersections of the five dimensions of user, time, terminal, channel and interactive process were represented by a permanently established network of interconnections in the IDMS database. A reservation was represented by marking appropriate nodes. Airline seat reservation is a simple subset of this problem.

In a system of this complexity, it is very difficult to anticipate the resulting performance, as the time to search along the various axes of the stored data depends critically upon details of the data structure. The reservation system required very short response times, or the time needed to do reservations would approach the total terminal time available.

The model also included a reservation dialogue by which the user would obtain information about unreserved facilities and

make reservations, and various data management dialogues. The dialogues combined minimum key depressions for experienced users with prompting for required data for inexperienced users, and allowed minimum repetition of data when some part of the input was found to be invalid.

The whole prototype system was implemented, and extensive performance and usability test and adjustments carried out, in under a year of solo effort—probably less than the effort required to write the specification for the system under conventional project management rules. The speed of implementation reflects not so much the skill of the implementor, but the excellence of the tools provided by ICL.

Hofstadter[1] has declared a prototype principle: 'The most specific event can serve as a general example of a class of events', and goes on to say, '. . . specific events have a vividness which imprints them so strongly on the memory that they can later be used as models for other events . . .'.

The principle may be stated more positively from the system designers point of view as: 'When you cannot think exactly what to do, do something' (it works just as well for jazz saxophone as it does for system design). The alternative, which may be called the Micawber principle, leads to a very low rate of learning and development.

Yours faithfully
JEFF REEVES
Computing Service
University of Southampton
UK

Reference

1. Douglas R. Hofstadter, *Gödel, Escher, Bach: An Eternal Golden Braid.* Penguin Books, p. 352 (1980).

Dear Sir,

Checking the reproduction of programs

Dunham[1] mentions the quandary of choosing between the cost of distributing programs in machine readable form and the error inherent in copying printed programs.

The solution proposed by Jacobs,[2] although better than nothing, is not effective in detecting letter inversions for example, a common copying error, nor does it aid much in locating an erroneous line in a long program. To propose a standard method acceptable to everyone is, in any event, vain.

When sending out a program listing, I use a small additional program which appends a running check-sum to the end of each line of the main program. The small program which generates the running check-sums is also sent.

The concept of a running check-sum provides the means of rapidly finding a line in error as all check-sums past the (first) line in error will be different from the original. The check-sum used is a simple analogue of the cyclic redundancy check.

Below is an example in FORTRAN 77. Program A is an erroneous copy of Program B. Program B is the correct program which produces a listing with running check-sums when a source program is read as input.

The extra programming compared to Jacobs' suggestion is minimal, but the result provides at once a better check-sum and a more efficient method for locating a line in error in a long program.

```
      PROGRAM CKSUM                                                      72
C                        PROGRAM A:     CONTAINS AN ERROR                72
      CHARACTER ABET*49, LINE*72                                          3
      INTEGER OUT, SUM, STRING                                          235
      DATA ABET /' ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789=+-*/(),.$'':'/  163
      DATA IN, OUT, NSHIFT, MODULO, SUM, NONSTD, STRING                 194
     +   / 5 , 6 ,   67 ,   487 ,   0 ,   0 ,     0 /                  449
      WRITE (OUT,'(''1'')')                                             271
C                                                                       271
    1 READ (IN, '(A)', END=4) LINE                                      131
      IF (LINE(1:1).EQ.'C') GO TO 3                                     387
      DO 2 J=1,72                                                       384
      K=INDEX(ABET,LINE(J:J))                                           273
      IF (K.EQ.1 .AND. STRING.EQ.0) GO TO 2                             386   √
      IF (K.EQ.84) STRING=1-STRING                                      252   x
      IF (K.EQ.0) NONSTD=NONSTD+1                                       266
      SUM=MOD( SUM*NSHIFT+K , MODULO )                                  228
    2 CONTINUE                                                           60
    3 WRITE (OUT,'(1X,A,3X,I4)') LINE, SUM                              419
      GO TO 1                                                           148
C                                                                       148
    4 WRITE (OUT,5) NONSTD                                              328
    5 FORMAT(//,1X,I7,' NON-FTN77 CHARACTER OCCURRENCES',/,'1')          95
      STOP                                                              145
      END                                                               457


      0 NON-FTN77 CHARACTER OCCURRENCES
```

```
      PROGRAM CKSUM                                                      72
C                        PROGRAM B:     THIS PROGRAM CORRECTLY           72
C       GENERATES A RUNNING CHECK-SUM AT THE END OF EACH LINE.           72
C       ANY SIGNIFICANT DISCREPENCY WILL MODIFY ALL CHECK-SUMS           72
C       AFTER THE INCRIMINATED LINE.   SPACES NOT ENCLOSED IN            72
C       QUOTES AND LINES BEGINNING WITH C ARE IGNORED.   THE             72
C       POSITION OF A NON-FTN77 CHARACTER IS SIGNIFICANT.                72
      CHARACTER ABET*49, LINE*72                                          3
      INTEGER OUT, SUM, STRING                                          235
      DATA ABET /' ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789=+-*/(),.$'':'/  163
      DATA IN, OUT, NSHIFT, MODULO, SUM, NONSTD, STRING                 194
     +   / 5 , 6 ,   67 ,   487 ,   0 ,   0 ,     0 /                  449
      WRITE (OUT,'(''1'')')                                             271
C                                                                       271
    1 READ (IN, '(A)', END=4) LINE                                      131
      IF (LINE(1:1).EQ.'C') GO TO 3                                     387
      DO 2 J=1,72                                                       384
      K=INDEX(ABET,LINE(J:J))                                           273
      IF (K.EQ.1 .AND. STRING.EQ.0) GO TO 2                             386
      IF (K.EQ.48) STRING=1-STRING                                       12
      IF (K.EQ.0) NONSTD=NONSTD+1                                       426
      SUM=MOD( SUM*NSHIFT+K , MODULO )                                  291
    2 CONTINUE                                                          306
    3 WRITE (OUT,'(1X,A,3X,I4)') LINE, SUM                              312
      GO TO 1                                                            38
C                                                                        38
    4 WRITE (OUT,5) NONSTD                                              174
    5 FORMAT(//,1X,I7,' NON-FTN77 CHARACTER OCCURRENCES',/,'1')         231
      STOP                                                               35
      END                                                               385


      0 NON-FTN77 CHARACTER OCCURRENCES
```