THE PROBLEM OF DEBUGGING THE LARGE ON-LINE SYSTEM

Dr. K. I. McKenzie

Computer Analysts & Programmers Ltd.,
1, Station Road,
Reading,
Berkshire.

Abstract:

Methods of organizing the de-bugging of large on-line programs are discussed, with particular reference to a dedicated commercial system.

Rapporteurs:

Mr. M. J. Elphick
Mr. D. Appleton

Dr. McKenzie started by remarking that he believed that a solution to the problem of constructing a large system, completely free of errors, was impossible at present. Our objective should rather be to define an acceptable level of reliability, and to develop techniques for maintaining this level. He felt that the subject could not be approached in an academic way - it should be taught rather as engineering practice is in other fields. Students should be taught, not only the technical aspects of program testing, but also the management of such projects - how to create, plan and meet schedules.

The development of techniques for program testing in general had lagged behind that of program construction, since 1955 at least, when the two tasks were comparable in effort. Programmers currently adopted a 'head in the sand' attitude towards program testing: the inexperienced programmer is tempted to think that exhaustive testing is always possible and when this proves impossible he goes to the other extreme of accepting the results of a few test cases as proof of correctness. This approach is clearly not adequate for the construction of large computer systems.

One problem has been found to be that of establishing a common vocabulary in discussion between programmer and manager. The simple question - 'does your program work?' - is not usefully answered with a straightforward 'yes' or 'no'. It had been found useful to adopt working definitions of a number of terms used in this context, of which the following are examples:

coded	-	source code written;
assembled	-	syntactically correct and complete;
exercized	-	every instruction executed at least once (can be quantified if not complete);
tested	-	works on agreed test data (not completely proved);
tried	-	performs reliably in a simulated environment;
viable	-	performs reliably in the live environment.

The requirements of a satisfactory 'test bed' for a large system are:

- a suitable system structure (separation into modules, linkage conventions, etc.), which must be developed with testing requirements in mind;
- a linkage editing system (as provided under OS/360);

- an adequate and flexible macro language.

The features provided by such a system should include:

- incorporation of 'drivers' for programs;
- test data files;
- simulation of real system hardware;
- simulation of special actions (e.g. I/O error conditions);
- easy transition to the real system.

The speaker estimated that 25% to 50% of the total effort might go into producing code which would not be used in the final 'live' system.

The description of the construction of an actual system followed. This was a dedicated commercial system (handling branch accounting transactions for a bank), requiring the multiprogramming of a number of collaborating tasks. Since the standard operating system used (OS/360-MFT) was not efficient in this context and the alternative MVT version was expensive in its use of core storage, the decision was taken to write a set of 'central service modules', resident with the supervisor, which would provide the functions of work scheduling, device handling, and an access method more efficient than 'indexed sequential' for this application. The application programs involved some 60,000 lines of COBOL source text, while the control programs required between 15,000 and 20,000 lines of Assembler code, divided into approximately 50 functional components, each component comprising on average ten modules. The design technique used was to define the system in a high-level language, and translate by hand to Assembler code. Testing had to be carried out in a batch environment via an operator service, until the final stage of system testing.

The speaker discussed examples of the system structure. Figures 1-3 demonstrate the use of a set of macros. LEVEL (figure 1) is a typical entry macro, providing parameter control of the test facilities included in the system, while PASS and XSVC are for direct and indirect transfer of control to routines. XSVC gives an indirect link via a permanent data area to central service modules, each multiprogrammed task having a copy of this area in its own partition.

For testing (figure 2), the special macro XTEST is used, again with indirect linkage via the permanent data area to the appropriate one of two routines. This permits the testing to switch between real and simulated

hardware control routines, and is used in conjunction with the macros XFIND and XSET to effect the changeover. This simple scheme goes some small way towards permitting a flexible transition from the simulated to the real system, but depends vitally on the system structure used. Among other facilities of a similar nature, the USEREXIT macro (figure 3) gives the option of calling by name special test routines not required in the live system. The SETUSER macro in the driver program sets up a correspondence in the 'User Exit List' between the name and the routine to be called (A and RTN in the example). Once TRTN is checked, the SETUSER macro call is removed and USEREXIT A will then have no effect.

Another problem which the speaker felt was not yet solved was that of inserting revised routines into the 'live' system in a safe way. The method adopted in the system described was to provide a method of 'enhancement control', to allow the orderly replacement of specified modules. This is illustrated by figure 4, in which the second parameter of the LEVEL macro allows replacement of one program module by another under control of the corresponding bit in an 'enhancement mask'. The second module may alternatively be used to augment the first, rather than bypass it.

This technique provides the ability to deliver a working system, with a controlled way of inserting improvements (planned previously) at a later date. Dr. McKenzie felt that it should not be too difficult to devise a mechanism for inserting modifications which were not foreseen in the original design: the difficulty was in providing such a facility without an unacceptable drop in efficiency.

In planning for the construction and testing of such systems, the methods C.A.P. used were not particularly advanced. However, they did have a fairly rigid way of laying out a series of plans for testing, illustrated by figure 5. This demonstrates the conventions used to distinguish between testing plans, requirements (for carrying these out), and the targets to be achieved. Although the terms used are not very precise, these plans enable the progress of testing to be controlled. The speaker remarked that PERT techniques had not been found very successful here.

Notes on the Figures

Figure 1: a structure on which the testing facility to be described can be implemented. Note transfer via address of routine stored in data area.

Figure 2: an example showing the structure of the test system. The routine used (for example) to write messages to the operator can conveniently be changed as testing proceeds.

Figure 3: a pass to TRTN will execute normally unless the user has 1) set the correspondence A, RTN in the driver, and 2) included the macro USER EXIT A in TRTN. In this case, control is transferred to RTN, which is a subroutine in the testing facility, when the USER EXIT A macro is encountered.

Figure 4: if the bit specified in RTN1 by ENH (BIT, RTN2) is set from the console, RTN2 is called. This feature may be used either to include additional modules or to replace existing ones.

Figure 5: sample scheme for testing constituent routines of code in an efficient manner. XY will have been the end circle of 01DM.

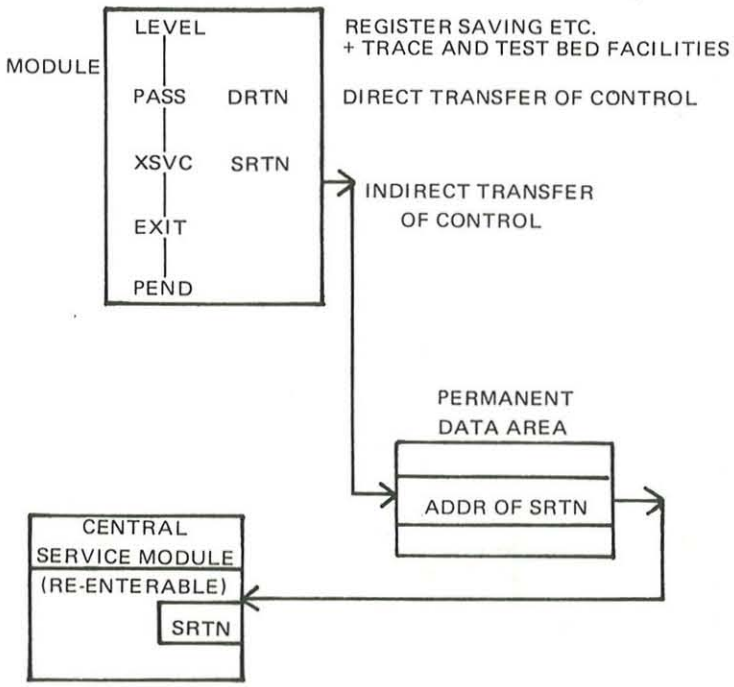

Figure 1 AN EXAMPLE OF A SYSTEM STRUCTURE

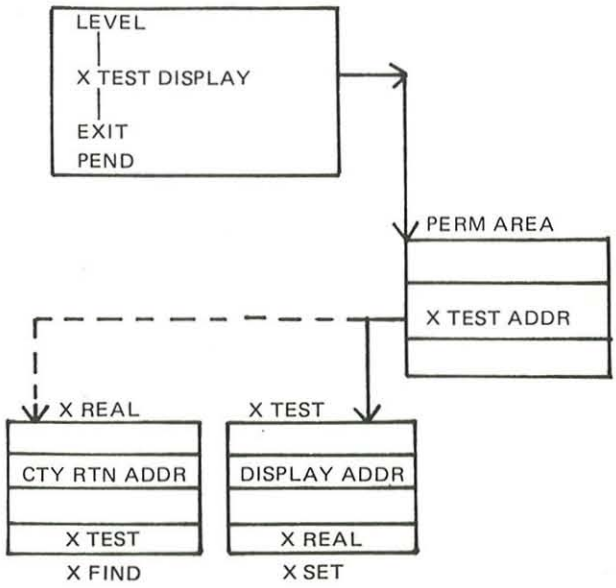

Figure 2 TEST SYSTEM STRUCTURE

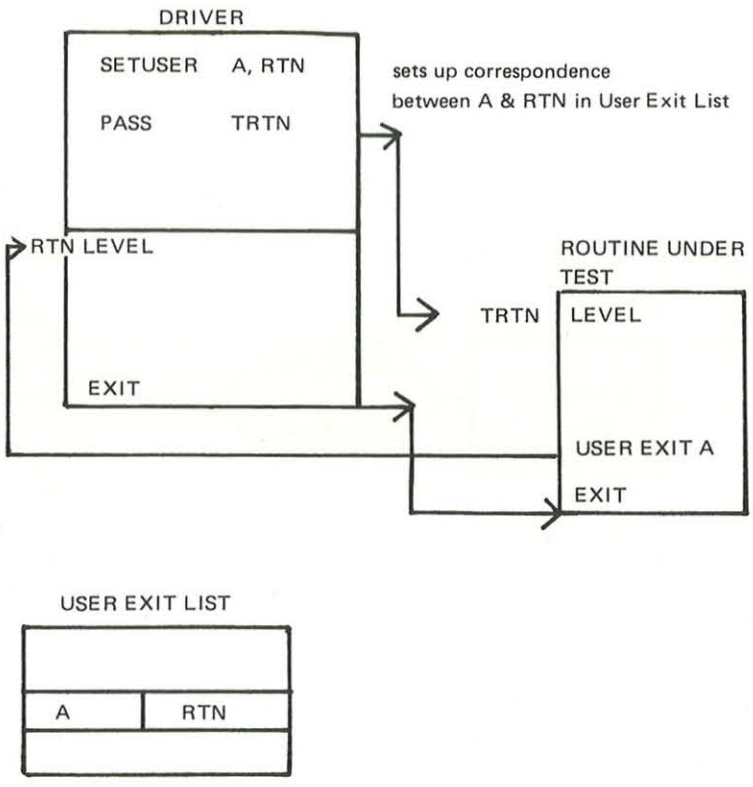

Figure 3 THE USER EXIT FACILITY

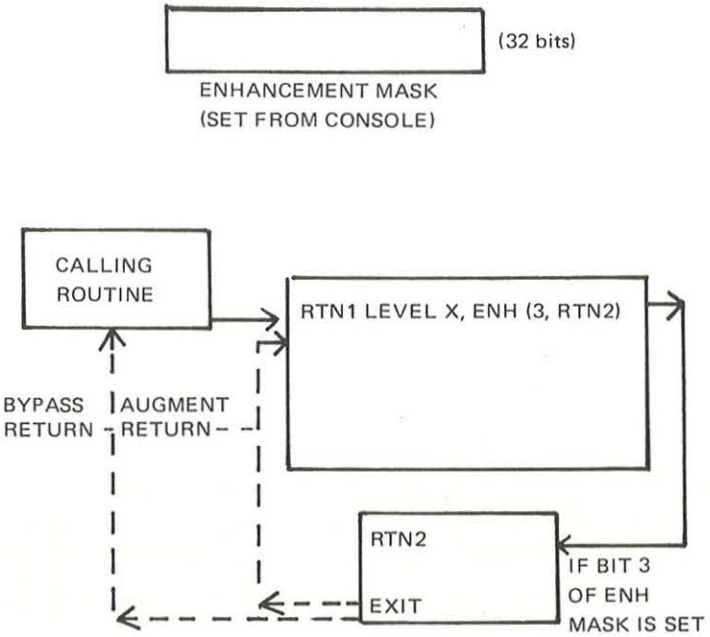

Figure 4 ENHANCEMENT CONTROL

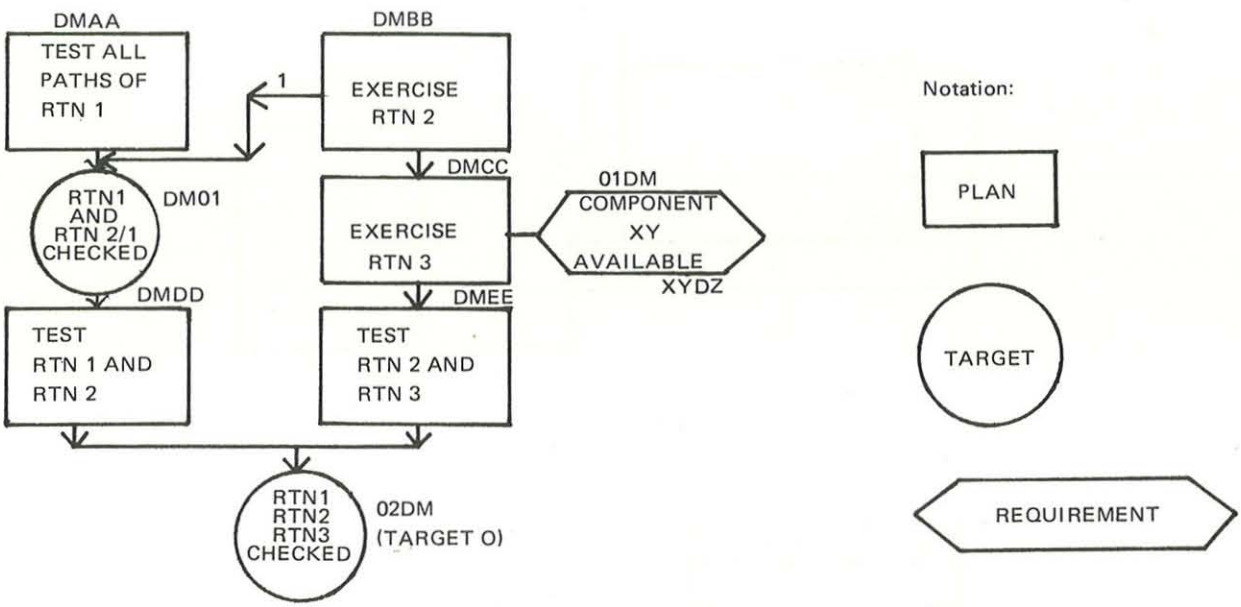

Figure 5 PLAN DM

DISCUSSION

Professor Coffman referred to Dijkstra's work (Dijkstra, 1968) on the construction of programs guaranteed a priori to be error-free, rather than on an a posteriori process of validation. Dr. Griffiths commented that in trying to put some of these ideas into practice with intelligent students the most important factor was 'making them write programs so that it is obvious that they do what they are supposed to'. Dr. McKenzie agreed that Dijkstra's work was very important, and that this more theoretical approach definitely ought to be taught, as well as the engineering one.

Professor Dennis suggested that once a program became operative the risk of loss of information or invalid results made the process of piecemeal modification of the system undesirable. One might get round this by creating a new system, sharing with the current version those modules to be retained and incorporating all new modules to be inserted. The problem then becomes that of transferring control to the new system, and the solution requires the concept of a 'mapping' of the current state of the old system into an equivalent state of the new one. Dr. McKenzie replied that there were in fact two problems - the actual transfer to the modified system, and (more practically) the lack of the required hardware resources to run the two systems concurrently.

Professor Hoare initiated a lively discussion of the relative merits of linkage editors versus faster compilers in the process of system modification. Dr. McKenzie felt that for a given machine, the re-linking of existing and newly compiled modules should always be more efficient than recompiling the whole system, while Professor Dennis said that if one followed his approach one would want to do this anyway.

In reply to a question from Dr. Scoins on the problems of communication between individuals in a team making alterations to a large system, Dr. McKenzie admitted that on the project he had described only two people in the team knew enough about the system to vet proposed changes. Although this simplified the communication problem, it was a strain on the persons concerned. What was really wanted for this was an on-line Management Information System.

Reference

Dijkstra, E. W. (1968) 'The Structure of the "THE" Multiprogramming System' Comm. ACM, Vol. 11, pp. 341-346.