System software real-time testing aids

W.E. QUILLIN
Plessey Radar, The Plessey Co, England

1. General methods of software system build-up

1.1 Design

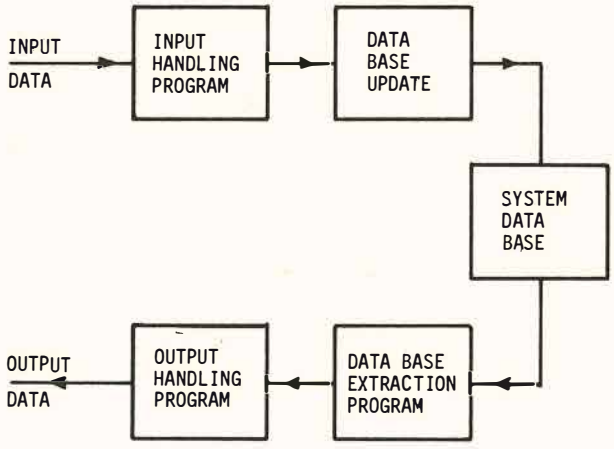
Before the use of testing aids can be considered it is helpful to consider the way a system is built. The system design is a top-down process, which takes the system from an overall specification to the design of individual program modules, which will be coded, tested and integrated to build the complete system. The design process also leads to the choice of suitable hardware.

1.2 Off-line testing

System implementation is a bottom-up process, starting with the testing and integration of the lowest levels of the modules. Initial testing will be in an off-line environment, in either a free-standing computer of the type for which the object code is intended, or in a computer of a different type, generally larger, which has a suitable emulator program. It is important that this off-line testing stage should be as thorough as possible, and remove all the coding bugs and design errors it can possible catch. A single free-standing computer is usually much easier to obtain and control for testing purposes than a complete on-line system. Some routines can be very extensively tested in this manner, others are much more difficult to off-line test, especially if they control complex interface equipment with real-time constraints. Not only single modules, but suitable collections of related modules, will be tested off-line. Input and output modules of these collections will be replaced by dummies, and assistance in the testing is provided by a set of off-line testing aids. The majority of faults which escape detection at this off-line testing stage will be interface errors and errors due to the constraints of real-time working.

1.3 On-line testing

For each package of modules there comes a time when either it is not possible to carry out any further off-line tests, or to do so would involve the production of an unacceptable amount of test program or test data. At this point on-line testing is started, and it is at this time that system testing aids are required. Like the off-line tests, the on-line testing starts with as simple a system as possible. The code is tested in its final object computer, but not all the other system computers and the full input-output facilities will be provided initially, see Fig. 1. Part of the job of the on-line test aids is to cover up the lack of these facilities and make it appear to the code under test that it is in its full system environment. The testing aids are primarily intended to help test the software, and leave it to a pre-test checkout or hardware detection routines included in the package under test to find any system hardware failures. The problem with a real-time system is that it is not designed to be capable of stopping and then being restarted. Hence the use of freeze-points in the code to trap errors is generally inappropriate. The testing must take place with the system in operation and it is highly desirable that minor changes can be made, whilst the system continues to run.



TEST 1 INPUT HANDLING PROGRAM
TEST 2 OUTPUT HANDLING PROGRAM
TEST 3 INPUT HANDLING PLUS DATA BASE UPDATE
TEST 4 OUTPUT HANDLING PLUS DATA BASE EXTRACT
TEST 5 COMPLETE SYSTEM

Fig. 1

2. Requirements for system testing aids

Testing aids are required to fulfil the following functions in a software system under development.

1. To inject data into the system under test. The simplest form of data injection required is the addition of items to stacks, queues and lists in the system, to check that the processing of this data is initiated and performed correctly. Such requirements generally arise because the system under test is incomplete. The need also arises sometimes during testing as it becomes necessary to exercise a suspect program on a more rapid basis than would be the case in normal use. Another type of data injection is the on-line amendment of contents of common core store and computer core store. This gives test personnel a method of controlling the running system, temporarily masking faults to enable other faults to be found, and, if they are not careful, completeley wrecking the software under test. It is a facility which must be used with caution, particularly when changing computer core store contents if the computers have a relocatable loader. The system testing software should enable tasks of the system on-line operating system to be initiated or suspended during the test by suitable data injection.

2. To extract data from the system for subsequent analysis. Once again the simplest case is extracting data from a stack, queue or list. There are two cases here, one where the queue or list forms the end of the process under test, i.e. the program which would usually empty it is not yet included. In this case the testing aid must remove entries at a suitable rate to prevent unwanted list or queue full indications. The second case is more complicated, where the list or queue is an intermediate point in the software under test. In this case the test program must look at and record entries but not remove them. The sampling rate should be rapid, so entries are not missed, but in practice if there is no designed phase relationship between the test programs, the filling program, and emptying program this is generally impossible as the occasion will arise where a list is read immediately after an entry is added and the monitoring software has no time to catch the entry. The debugging aids must also be able to observe more general system data, such as models showing current system configuration, current state of world models in which the system has an interest, operator and data link injected data, fault reports, etc.

3. To vary the real-time working of the system. Requirements arise during testing when it is required that the system should run faster or slower than real-time. Difficult faults may be found easier in a slow running system – the ultimate in slow running a system is to be able to 'One-Shot' it, and the system testing aids should make this possible, in conjunction with the on-line operating system. Other system faults may require the system to run faster than real-time, this being especially true of faults which require assistance of engineers using instruments, e.g. oscilloscopes. Testing, both hardware and software, is designed to ensure that such faults are rare, but they cannot be completely eliminated. A facility should also be provided to enable part of the program under test to be continuously repeated, as an aid to finding intermittent and other seldom occurring faults.

4. The system testing forms part of the total documentation. This is required to give a record of the testing methods used and to assist in the tracing of faults which occur during use and during subsequent software or hardware modifications. The system testing software can assist this documentation task by giving a print-out in a suitable form for incorporation in the documentation.

5. Frequently during testing the designed method of system start-up and shut-down is inappropriate due to the need to run with parts of the system incomplete and the desire to reduce testing time when possible. The system test software should enable various start-up and shut-down procedures to be followed as appropriate to the test being conducted.

3. Examples of testing software successfully used on on-line systems

The following examples consider two on-line test programs written and used by the Plessey Co. for testing on-line software. They are currently being modified and integrated, but for the purposes of this paper they are considered as individual programs.

3.1 Electronic data display monitor

This program was written to run on a system which has a number of operator positions which include alpha-numeric keyboard inputs and electronic data displays (or VDUs), onto which the system can write alpha-numeric characters, as output. The same design could be applied to other I/O media, e.g. keyboard and teleprinter or lineprinter. This would help the output to be of assistance for documentation reasons, but it can lead to the problem of the system testers disappearing under a mountain of paper whilst they search for a sheet which is relevant to a problem under test. The program was in fact so

arranged that inputs can be accepted from paper tape and the computer control console as the positions' keyboards. In practice these first two options have rarely been used.

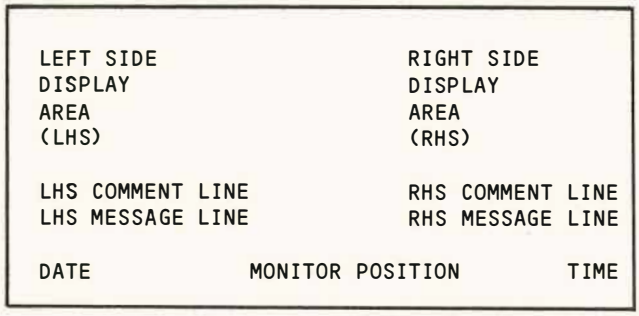


Fig. 2 Basic display for EDD monitor

3.2 The basic display is shown in Fig. 2. It requires a position with at least a 10 line display facility. The program will operate if using a position with only 8 lines (in one case some positions had only 8) but the Date, Monitor Position and Time Displays are then not available. The position in use is set up at the start of a trial and can be changed if required during the testing period. Each side of the display is treated independently and on each side there can be displayed up to six words of eight or twelve digits. Data is loaded into the display formats from the top and if a smaller number of words than six is required the lower data lines will remain blank.

3.3 There are four types of data messages:

i Display up to six words from a common core store.
ii Display up to six words from a computer store.
iii Inject up to 6 words into a common core store.
iv Inject up to 6 words into a computer store.

Data is set up on the monitor in octal. In the case of injections to core store, the operator must confirm the data and then inject, giving ample chance to remove an error. Repeated injections of the data set up are possible.

3.4 There are also four possible auxiliary messages:

i Add 'nn' to a given address (nn is a two-digit octal number).
ii Subtract 'nn' from a given address.
iii Continually update displayed data.
iv Freeze display.

The first two apply to any of the Data messages, the second two only to Display Data messages.

3.5 There are also four clear and erase messages:

i Clear a whole display.
ii Clear right side of display.
iii Clear left side of display.
iv Erase last character injected.

3.6 There are comment messages which are given to the operator upon a fault condition.

i Wrong Key – Try Again.

This is displayed upon operation of an illegal alpha-numeric key. It is cleared on a valid injection.

ii No Data message displayed.

This is displayed if an operator attempts an Auxiliary Data message when no Data message has been injected.

iii Common core address invalid.

If the full address field is not used for common core store addresses and an out of limit request is made, this message will be displayed.

iv Faulty Transfer.

After an inject to store message the program sends the data, then reads it back as a check. This message is displayed if the check fails.

v Inject Inhibited.

It is sometimes necessary to inhibit the injection of data, for example during system trials. This inhibit is controlled from a computer console key. The above message is displayed if the operator tries to inject whilst this key is operated.

vi Computer Not Available.

Whilst transferring injections to or from another computer the program expects a confirm reply. If this does not arrive in a predetermined time, the above message is displayed.

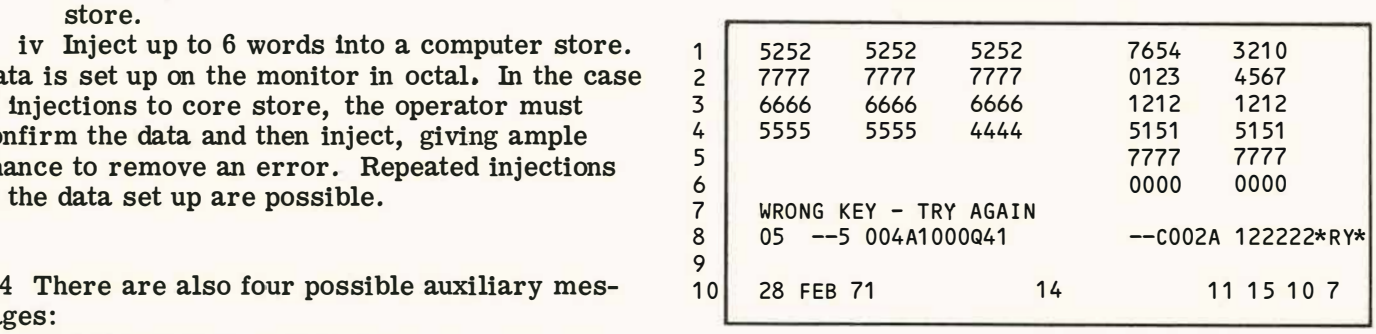


Fig. 3 Sample format of EDD monitor

3.7 Figure 3 shows a sample format. The left side display "--S 004 A 1000 Q4 I" means inject the four words of data into store 004, starting at

address 1000. The 05 is the number of times this data has been injected. The operator has operated an invalid key. The date is 28th February, 1971. The right side display message "--C 002 A 122222*" means display 6 words of data from computer 002 starting at address 122222. The Auxiliary Data Message RY* means continually update the display. The time is shown as 11.15 10.7, i.e. 10.7 secs after quarter past eleven.

3.8 Experience with this program has shown that it makes it possible for an experienced system test team to locate errors by working in an interactive fashion with the running system. The use of a display enables many data items in computers and common core stores to be scanned rapidly, to find for example where a chain of events breaks. Data changes of $\frac{1}{4}$ sec can be detected on the display enabling entries in queues and lists to be seen between addition and removal.

4. SCOT program

The second system test aid program to be described is called SCOT (System Computer Online Test) and covers testing functions described in paragraph 2 which are not covered by the EDD Monitor program, such as adding data to queues, giving output suitable for documentation and system start-up. Whereas the EDD monitor program is designed for searching for faults and obtaining small amounts of data (by writing down display contents or photography) the SCOT program is used to obtain large volume outputs.

4.1 The following description of input messages shows the program's capabilities.

i Specify common stores in use for the test. The address numbers of the stores to be used are input.
ii Place data in common stores. From one word up to a complete store may be set up.
iii Extract all entries from a queue and reset control word. The entries extracted are printed out.
iv As above, for a list.
v Compare Data in a specified area of a common store with its previous contents. Any differences are printed out.
vi Add an entry to a queue.
vii Add an entry to a list.

Write directives (ii), (vi) and (vii) must be preceded by a time-tag which can range from $\frac{1}{4}$ sec upwards in steps of $\frac{1}{4}$ sec. Extract directives may be time-tagged. If they are not, they will be executed every $\frac{1}{4}$ sec. A choice of time sources is provided – the program will use a system clock if available, if not a programmed timer can be used – which is less accurate.

There are a number of program usable control keys on the computer consoles. Keys one to twenty can be used to switch individual directives on or off if they specify the key number.

As well as pre-loading the common core stores the program can be used to start system computers in any order by sending specially coded messages to them, whilst they are in a start-up state. Queues, Lists and Data areas can be referenced by a CORAL name instead of the store number and address. This is arranged by a declaration of the form

⟨name⟩ * store number store address

where * is either Q, L or D for queue, list or data area.

```
S   2,  3,  15
K1   T   1   47.5
WDS   3   A   1250
C 1234   5670   1234
C 5252    5252    5252
T3   30.0
EQS  4   A 1000
```

Fig. 4 SCOT input message example

4.2 Figure 4 shows an example of a SCOT input message. First the stores in use for the test are declared, in this case 2, 3, and 15.

The next message is to write to store 3, address 1250 the two words given in octal. This is after 1 min 47.5 secs, and as KI is specified it will only be obeyed if computer console key 1 is operated.

The next message, which is independent of the console keys as none is specified, should be actioned at $3\frac{1}{2}$ min after the program starts. It is to extract data from a queue in store 4, address 1000. In fact, it will result in an error when it is read into the computer, as only stores 2, 3 and 15 have been declared for the test.

4.3 Figure 5 shows the form of a SCOT output. First the start time is output if the program has access to the system clock. Then the system stores in use are shown, not exactly as depicted here as the present version outputs the store availability table built up as a result of the store declaration input. However, this example has been simplified to make it more clear, and avoid a need for system table layout knowledge.

The next output message echoes the input which was to write to store 3, address 1250 the two octal constants shown, as a check that this action was completed.

```
START TIME    12   45   52   25
SAL 02
SAL 03
SAL 15
Str 3    Adr 1250
C 1234   5670   1234
C 5252   5252   5252
E min 3 sec 30.0      Str 3 Adr 1000
C 0003   0003   0401    cnt 3 tai 1 hed 3
C 0357   4732   0101
C 0
C 3716   5125   3076
C 1327   0312   7766
```

Fig. 5 SCOT output message example

The next output message is the result of reading a queue in store 3, address 1000. Note that it is a queue of five words (four plus a control word) the maximum length being shown in the control word. The control word also shows where the head is (word 3), the tail (word 1) and the count of entries (3 in this case). These are output as decimal integers as shown.

4.4 Error reports from SCOT fall into three classes.

(a) Input faults.
- i Fault in common core store declaration or no stores declared.
- ii Fault in octal constant.
- iii Fault in number given for common store, out of range or not declared.
- iv Address fault.
- v Fault in decimal number.
- vi Fault in time given.
- vii Fault in name declaration.
- viii Name not declared.

(b) Store faults.
- i Initial check unsuccessful. The stores declared are checked at the start of the test, and should this check fail this message is output, along with the data sent, data received and the store number and address.
- ii Transfer of data unsuccessful. Data written to a store is read back and checked. If the check fails, this message is output along with the four items as in (i).
- iii Routining fault. Every $\frac{1}{4}$ sec a routining pattern in a reserved word in every store is checked. A fault causes this output along with store number and data returned.

(c) Access faults.
- i Queue full.
- ii List full.
- iii Queue locked out too long.
- iv List locked out too long.
- v Computer failed to start.

The time is given along with the above five fault reports.

4.5 Experience has shown SCOT to be a valuable system testing aid. It is particularly good at eliminating interface errors before system integration tests. To achieve this, the programmer who is writing say a program to extract data out of a queue and process it will get a programmer whose code would normally help fill the queue to provide a test tape, thus ensuring no misunderstanding.

The hardware tests of common core stores were included in SCOT as a 'long stop' to help show any faults which may arise during a test. These checks have shown faults during test runs, but these have generally been due to manual equipment allocation errors, not hardware, when a store has been out of system when required or removed in error during a test. A system is more prone to such errors during build-up time as the tests do not often require the total system and other activities are allowed to use the spare equipment.

5. Hardware assistance with system testing

The constant problem during testing is one of data collection – finding out what state the system was in just before it faulted. Up to now this assistance has usually been arranged by special software, and this paper has considered in detail two programs which have been used to achieve this. It is also possible to arrange for the system hardware to collect data to assist in on-line testing. This can be by hardware built-in to the the system or special-purpose test equipment which can be attached. These alternatives are now considered.

5.1 Facilities built-in to the computer hardware

In the case of real-time computer systems there are a number of ways the hardware design can help the system testers. Historical registers and test driver facilities are examples of these.

- i Historical registers.
  The computer contains a group of, say, sixteen registers addressed sequentially so they form a circular queue. When an instruction involving a single store operand is obeyed, three values are placed in these historical registers:

Instruction Address Register, Absolute Value.
Instruction.
Store Operand Address.

The third item is omitted if the instruction does not involve an operand address. Hence the last six or seven machine instructions obeyed can always be found in the historical registers. Their contents can be frozen by either a fault interrupt or by program control, and their contents then extracted.

ii Test driver facilities.

Special interface facilities can be provided so that the computer may be test-driven by another computer (of the same or a different type) which can access its registers and core store. This enables a trace of the computer's activities to be obtained during real-time working, when a software trace facility is inappropriate due to the slowing down of run time (typically by a factor of 80).

5.2 System attachments

System attachments can be obtained for data recording. These are generally similar to the historical registers described above, but they are portable and will usually interface with any convenient points of the system, e.g. store data lines, store address lines, interconnection buses, peripheral devices. Data is recorded in special registers or on magnetic tape for subsequent analysis off-line. They are particularly useful towards the end of system testing to measure use factors of connection channels and look for system bottlenecks. Some are equipped with an address comparator which can be used to show, for example, the amount of time spent in the operating system routines. Once experience is gained patching such monitors into a system they can quickly provide detailed information on suspected fault areas without requiring special on-line software.

6. Conclusion

Experience has shown that time and money spent on providing system testing aids has been essential to the testing of on-line system software, from the start of first on-line tests up to final system trials. Even after a system becomes operational some faults will continue to occur and the test aids have a continued value.

As the cost of hardware decreases and computer designers pay more attention to software needs, more assistance is likely to be given by special circuitry to the on-line testing of computer systems.

Discussion

Q. Do your debugging aids run as part of the user's program in real time?

A. Yes. It depends on having enough capacity in the computer, which we have.

Q. Is it possible to make the program run more slowly than normal?

A. Yes, the rate can be controlled.