SOURCES OF INFORMATION

IMPORTANT PRINCIPLE

MESSAGES PURPORTING TO GIVE INFORMATION
MUST BE
COMPREHENSIBLE !

Like this ? (from CMS on the University Computer Centre's IBM4341, 14 August 1987. The IBM4341 was replaced with a Silicon Graphics machine in 1989.) (The first two lines are reports from the Archiver programme, about which you can read more in the *ARCHIVING* chapter.)

```
Catalogue entry for CROAK INPUT1 has been Altered
Catalogue entry for CROAK INPUT2 has been Altered
DMSITP143T PROTECTION EXCEPTION OCCURRED AT F65D7C IN SYSTEM ROUTINE
TYPLIN, RE-IPL CMS.
DMKDSP450W CP ENTERED; DISABLED WAIT PSW '00020000 50F7F4FA'
```

Well, perhaps not quite like that. There, some important information was completely obscured in a rather unintelligible message. (It's true that the example is from 1987; fortunately, things are much better now. Aren't they ? No[REQ17].) It is usually true that when something goes wrong there is enough information in the system somewhere to determine fairly accurately what it is, so comprehensible error reports expressed in terms of what the affected person can see of the system are usually possible in principle. In practice, it can be hard to collect the information needed for the diagnosis, but that doesn't mean you shouldn't try.

Error reports are only one sort of information which you should be able to get from an operating system. There are at least two other sorts – facts about the system itself, and news. Whether you *should* also be able to get help from the system is possibly not quite so obvious, if only because many – perhaps most – of the things which go wrong are not the system's responsibility. Our opinion is that the system should certainly provide a general help service, because there is no other way to maintain consistent behaviour.

Here we comment on four sorts of information which the system should in principle be able to offer to its clients. Our comments are indicative only, and we have attempted no detailed analysis of what could or should be done. Think about your own experience with computer systems : what sort of information would you have liked ? – could the system have provided it ? – and, if so, how could it best have been presented ?

HOW DO I DO THIS ? – the HELP system.

> *THEORY : at any point in your terminal session, you should be able to ask for help – and you should receive whatever help is appropriate in the context.*

There are several sorts of question :

- between instructions – WHAT CAN I DO NEXT ?

  This is the easiest sort of question to answer, for the answer amounts to a simple list of system instructions. Nevertheless, it's often not available. In a well equipped system, the complete list of possibilities might be too long to display, but it's usually only necessary to show the most popular set – experts probably don't need to look anyway.

- HOW DO I USE X ? – where X is a piece of hardware or software.

This is also a factual question, and can be answered directly from a file; the two necessary tricks are, first, to be able to find the right file (not too hard provided that the questioner knows the right names for X), and, second, to get the right material in the file – it *isn't* really adequate to provide the system manual : text for displaying on a screen should be composed differently from text to be printed in a book.

- in the middle of an instruction – WHAT CAN I DO NEXT ?

  Again this is a factual question, but it's harder to answer; you need careful organisation to identify the right response, as the required answer might depend on just where you are in the instruction. It's possible, but rare.

- HOW DO I DO Y ? – where Y is some computing task.

  To answer this question, you probably need an expert system, though in practice a good keyword retrieval system performs fairly well. The difficulty is that Y, as presented, might not be a system instruction at all – "How do I get rid of a file ?".

These are all essentially textual answers, and there's some assumption that we're dealing with textual instructions. That's not because we're ignoring graphical techniques, but because there seems to be little effective system help in machines which rely on graphical interfaces. The original assumption was that you'd be able to see the full repertoire of what could be done on the desktop at any time; that stops working once you get more than eight or ten programmes, and doesn't include the system operations (change file names, delete files, etc.) which are carried out by purely manual operations – which is unfortunate, because they are the object of many questions. The Macintosh system, in its later versions, offers "bubble help", giving information about whatever screen object is currently under the screen pointer; this is an elegant approach to the provision of help, but it still depends on the information having been provided, and is as yet rather unselective (and often fairly trivial). Generally, the GUI convention that you have to be able to see something on the screen to do anything with it makes the provision of a general help system rather difficult.

Whatever sort of help system you have, one of the most serious challenges is to make it apply uniformly. As with the user interface conventions, it's necessary to establish standard ways of presenting and formatting help information, so that external suppliers of software can fit their products into the system. Whether with graphics or text, this is difficult; we guess that text is hard, but graphics is harder.

WHAT'S HAPPENING ? – the CURRENT STATE.

> *THEORY : You should always be able to find out what's going on in the system.*

We would prefer that our computing activities be constrained as little as possible, so if we want to find out something which the operating system knows (why the running programme couldn't find a file, whether the running programme is still running, whether we have any mail, what the time is) we should be able to do it.

This was often by no means easy on older text-based systems, presumably because they were built according to a single-programming computer metaphor. (Their designers wouldn't know that, of course – they just made the terminal do what they expected terminals to do.) A terminal was thought of as associated with one task, and one task alone, so the suggested ability was obviously silly, and couldn't be done.

Which is, of course, not so. The terminal has always played two rôles in the system (see *USING TERMINALS*), so the information was there if only anyone had thought to look. Some systems provided some items of information, typically in response to control keys; many Unix programmes would (and still will) allow you to pass an instruction through to the operating system by preceding it with "!", but that was unusual.

It's probably not an accident that Unix has always provided facilities to run several processes simultaneously.

With modern GUI screens, the situation is quite different. We are accustomed to seeing screens with several different things happening in different places, so this sort of information is comparatively easy to come by.

WHAT WENT WRONG ? – the DIAGNOSIS system.

> *THEORY : whenever a programme fails in any way, the system should be able to help you to find out why, and to put it right.*

The very first step in so doing is for the system to be able to tell us that something has gone wrong, and what it is. That's a significant statement, because it emphasises the point that if something has gone wrong then in some sense or other the system has lost control and there's no guarantee that it can find its way back to anything that can send us a message. A not uncommon phenomenon in early systems - and a not unknown phenomenon today - is the system which simply freezes for no apparent reason. A well developed exception management system is important in avoiding such events.

Once we've caught the failure, then at the simplest level we'd like informative (and comprehensible) error messages. It helps greatly if the messages includes some sort of address in the programme (source file line number, or – less good – code address) to say where the error was detected. (The CMS system *did* provide a HELP instruction which would give some further information about forbidding error codes like DMSITP143T and DMKDSP450W which appeared in the cautionary example at the beginning of this section, so it wasn't quite as bad as it looked – but rather few people knew about that provision, so in practice it wasn't much use.)

A quite separate problem is finding the correct words in which to report the error. Whoever is using the programme which went wrong is more likely to want to know that an item cost read from the inventory file is too large than that a floating point overflow exception has occurred. Unfortunately, the only entity which knows about the item cost and the inventory file is the running programme, over which the operating system has little control. If the operating system designer wants to make sure that some error message is produced, there might be no alternative to reporting the floating point overflow exception.

We mentioned this difficulty in the chapter *DEFINING A SYSTEM INTERFACE*. It's a problem which cannot be avoided in any layered system, where one part of the low level software might be used by many different parts of the high level software. The only approximation to a general solution which we know is to build error reports into software at all levels, and to have some means to pass on at each level of procedure call (or equivalent) some indication of whether the calling procedure will handle error reporting. Then a procedure which can report errors and finds that the procedure which called it cannot handle errors should make the report. In any case, of course, some indication of the type of error found should be returned. It's a cumbersome solution to the problem, but does work provided that the implied conventions are generally observed.

Other people are involved too, of course; so for programmers there should be software to produce and interpret memory dumps, various tools which allow you to observe a programme as it runs, and so on. One can debate whether or not these facilities should be the responsibility of the operating system.

WHAT'S NEW ? – the NEWS system.

> *THEORY : everyone using the system should be kept informed of changes in hardware, software, administration, ….…*

This is obviously a very sensible principle, but it's extraordinarily difficult to manage. A major difficulty is that if you send people messages through the computer system they will ignore them, lose them, or forget them. Messages on paper don't necessarily fare much better. Many systems provide facilities to display system messages during the logging-in sequence; people regard them as irrelevancies, and become accustomed to pressing the return key a few times automatically to skip past the message.

An alternative is to make available NEWS files for people to read, but then you find that most just don't bother.

The news files are the better solution of the two; but they must always be reliable, or they won't be trusted. You can send messages too, but they should direct people to the news files for details. The news files should also always contain something new, so that people don't get bored by reading the same messages time and time again. Unfortunately, what's new for someone who uses the system daily isn't the same as what's new for someone who only uses it once a month.

FINALLY –

All information systems should be integrated. That's HARD.

COMPARE :

Lane and Mooney[INT3], Chapter 14.

REFERENCE.

REQ17 : R.W. Lucky : "Fatal error number 27", *IEEE Spectrum* **34#5** 18 (May, 1997).

QUESTIONS.

How could you design a HELP server ? The aim would be a system-wide service, which would work uniformly for any component of the system which required it.

Can you put the four "sorts of question" into order of usefulness ?

From an examination answer : "... in a textual interface, it's easy to see what you've been doing, because the reports are still on the screen, and that's useful in interpreting errors". (Unix has a "history" instructions which will show you what you did even if the screen contents have gone.) Is that really useful ? Could you provide such a facility for a WIMP interface ?

Perhaps it would encourage programmers to produce good error messages if the system produced a bad one by default – so if the programmer does not supply an error message, the system can report "a floating point overflow exception has occurred, and your programmer was too careless to tell the system what it meant". Would it work ? How could you implement it ?
