An investigation into database resilience

P. A Dearnley

School of Computing Studies, University of East Anglia, Norwich NR4 7TJ

The concept of data base resilience is defined. A particular class of data base systems is examined for features which support resilience. The conduct of a series of experiments on an actual system is reported.

(Received July 1975)

Definitions

In normal English usage the word *resilience* is taken to mean *the power to resume original shape after compression*; in the context of data base management the term *data base resilience* is defined as *the ability to return to a previous state after the occurrence of some event or action which may have changed that state*. The particular state of interest is the logical content of a data base. The physical state of the data base may also be of interest if access to the knowledge of this physical state is allowed to application programs. However this knowledge is increasingly denied to application programs in the attempt to preserve data independence (CODASYL, 1971) or to simplify data access by removing the need for navigation (Date and Codd, 1974) and thus returning to a previous logical state is often sufficient. One type of event or action expected is an error in the operation of the system. Hence a particular type of data base resilience of interest is the ability to return to a previous logical data base after the occurrence of an error which may have damaged the logical data base. Terms related to the concept of resilience are: privacy, security and integrity. *Privacy* is defined as 'whether or not a particular individual should have access to a specific piece of information' (Conway *et al.*, 1972) and *security* as 'preventing unauthorised access to a file' (Dearnley, 1973). *Integrity* is defined as 'the safe-guarding by the system of information entrusted to it' (Wilkes, 1972). Thus data base resilience includes the ability to maintain integrity and a particular type of occurrence for which resilience is useful is the breaching of security with a subsequent unauthorised change to the data base. Similar definitions of these terms may be found in Browne (1972).

Provision for recovery

A conventional data base management system allows the data administrator to make various provisions for the recovery of a data base from erroneous change. The facilities provided usually include one or more of *physical copying, logical copying* and *quick-looks* (Fossum, 1974, Palmer, 1973). A physical copy of a data base is produced by dumping both data and control information on a track-by-track basis, from the original data base medium to the back-up medium without any regard for format or logical content. This operation is comparatively quick but it does not allow validity checking or data reorganisation. To remake the data base after an error has been detected the physical copy is restored from the back-up medium. The construction of a logical copy of the data base is performed with the knowledge of the relationships between records and of the format of records. While the data base is unloaded onto the back-up medium, validity checks can be performed and the opportunity occurs for 'garbage' to be deleted. During the reload operation the data allocated to an area may be reorganised to improve subsequent performance. Thus a logical copy may be preferred to a physical copy; however the unload and reload times for the logical copy will be longer than dump and restore times of the physical copy. The quicklooks method involves taking a copy of a page of the data base, on a direct access file, immediately before a run-unit makes its first change to that page. A limited number of such 'looks' are kept to provide a rapid but limited recovery mechanism. All these provisions for recovery consist of deliberately introducing data redundancy into the environment of a data base system which is usually designed to eliminate redundancy within the base itself. Some systems do not explicitly include in their design criteria the deliberate and consistent removal of duplicated data. In such systems different methods of recovery may be appropriate. Two of the types of systems which allow redundancy are the 'conventional' data processing system based on sequential files and the 'self organising' data management system. The conventional data processing system based on sequential files maintains 'grandfather', 'father' and 'son' master files which are updated in reconstruction mode thus allowing recovery to be effected by stepping back one generation and rerunning the system (Clifton, 1969). The self organising data management system creates and exploits duplication to provide cost effective alternative access strategies. Depending upon the nature of the redundancy and the location of the erroneous change to the data base, this duplication can also be used for recovery.

Self organising data management systems

This class of system is designed to meet the following criteria:

1. The system has the capability to determine and implement suitable structures for the files held in the data base. Structures are chosen with the object of minimising the total cost of known or predicted accesses.
2. The access strategy adapted is constructed by the system.
3. Any correctly specified task can be completed by adopting some access strategy and the user is given a cost quotation in advance.
4. The system can restructure or update files as a result of an accepted cost quotation, or, by observing patterns of usage and predicting that a different structure will be economically advantageous to the body of users.
5. The user is allowed to leave requests for tasks in the system in the hope that batching or structure changes will eventually reduce the cost of his task to an acceptable level.

The original concept of self organising data management systems is described in greater detail elsewhere (Stocker and Dearnley, 1973; 1974). A natural consequence of such design criteria is the complete separation of the user view of data (called the *folio*) and the physical structure of this data (referred to as *versions* of the folio); this is essential since the system is free to change the physical structure according to data usage. This separation means that to exhibit resilience a self organising data management system's base need only return to a logically correct state and a different physical state after an error condition is permissible. This logically correct state may already exist due to redundancy; that data originally used to answer one type of request and which is now damaged may also exist (to support data access for a different type of request) in another part of the base. The ability to dynamically select access paths (criteria 2 and 3 above) means that the self organising data management system can automatically make use of the alternative copies of the data if the definition of the damaged part is removed from the base; indeed the 'route finding' algorithm will select the cheapest access strategy if several alternatives exist. If the type of access for which the damaged version was originally used is frequent then the folio management algorithm may cause the version to be reconstructed by the system without the intervention of a data base administrator (see criteria 1 and 4 above). Hence the concept of a self organising data management system allows considerable potential for automatic data base resilience. To investigate this potential a series of experiments were conducted on an actual system.

Error detection

Before any recovery mechanisms can be employed it is necessary to be aware than an error has occurred and that the data base is damaged. Of the many types of errors that can occur one of particular relevance is the damage to a page of data on, or being returned to, the physical storage medium of the base. The presence of this damage may be noted when a block of data cannot be read back to primary memory or when it fails consistency checks after reading but prior to processing. Checks on field content and record format can be performed during the invocation of a run-unit and the data base administrator informed if errors are suspected. Checks on overall data base structure can be carried out during logical unload and reload operations or using a special structure auditing utility program (Stross, 1972). If a run-unit suspects error in the base then the data base administrator may, himself, use a special utility program to validate (or locate errors) in part of the data base. The experiments carried out on a self organising data management system use the audit program approach.

Test system

The system on which a series of experiments into data base resilience has been performed is a model of a self organising data management system. The model system implements design criteria 1 to 4 above and is a 'model' in the sense that it operates on a reduced size data base (only up to one million bytes) and does not deal with important but not crucial issues such as the elegance of the user interface. However it is a working model and contains the necessary route finding, folio management and version access routines in addition to a simple user query language. It is described in more detail in Dearnley (1974a). The model system is used to demonstrate that a viable data management system can be built along self organising guidelines and to obtain operational experience from such a system using a number of data bases under varying conditions (Dearnley, 1974b).

The system maintains three areas of data: a system directory, a system map and an area holding versions of user data. The system directory contains information describing the user view of data and the system internal view of data at a logical level. The user view of data is described in a record called the *folio header*; this record contains a simple tabular presentation of all the attributes known to a user for a particular collection of his information, and, a list of the versions containing the various physical representations of this information. For each such version there is a *version header* record describing the parts of the folio represented by that version, the record layout and the type of organisation employed. A further record, the *search addendum* is held for each version; this contains statistical data on the usage of this particular part of the representation of the folio. The system map is a series of blocks packed with the mapping between pages of the data base with relative addresses and the physical geometry of a particular direct access storage device. Parts of the map are loaded into main memory when the system is started up and only those parts which are changed to represent the creation of new versions or the deletion of old versions are written back to the system map area. The third area contains the versions of user data and temporary work files. The content is defined by the system directory, and, versions and work files are located according to the system map. Thus this area can be thought of by the user as the 'data base' and the two former areas are regarded as being private to the system. In the model system the directory and map occupy less than four per cent of the space occupied by the user data area. There is no appreciable duplication in the system directory or map but due to their comparatively small volume they can be cheaply backed-up with physical dumps. The user data may contain duplicate versions in different organisations or have fields duplicated in subrecords or records duplicated in subfiles depending upon the trade-off between the cost of the usage of existing versions and the cost of creation (or restructuring), followed by a preferred access method, on a new version. It is in these versions that one might expect duplication to provide some measure of automatic resilience without resort to back up of user data by physical or logical copying.

Experiments

The experiments are run in four separate phases. The first phase simulates the random damaging of the data base; this is followed by an audit phase which attempts to find the location of the damage. These two phases are repeated one hundred times to provide variety in the location and extent of the simulated damage. The result of the audit phase, if any, is in terms of the damage at a physical level, the third phase takes this result and determines the effect of this damage at the logical level as viewed by the system. The final phase determines the implications for the user view of the data base of the damage at the system's logical level. The overall sequence of operations and the use of the data base and intermediate files is shown in **Fig. 1.**

Corruption

Before attempts to damage the data base are made a utility program, REPORT, is run to provide information on the current status of the data base at both the user level (i.e. folio definitions) and at the system logical level (i.e. version definitions). To simulate damaging the data base a special program, CORRUPT, is loaded. CORRUPT selects, at random, one of the areas of the data base; the directory, the map or the user data. The frequency of selection of any one area is biased to reflect their relative sizes. A page, word and extent within the chosen area are then selected. These selections are, again, made at random. The chosen page is read into primary store. The undamaged copy of the page is written, along with the area name and page address, to the 'save file'; this allows the data base to be reconstructed prior to another experiment. The copy of the chosen page in primary store is corrupted by storing a sequence of random numbers starting at the randomly selected word address and continuing for the randomly selected extent. The corrupted page is then rewritten to the area. The page transfers requested by CORRUPT are routed to a special input/output package so as to avoid the validity checking and housekeeping portion of the data base system file handler.

Audit

A second program, AUDIT, is loaded to attempt to locate the

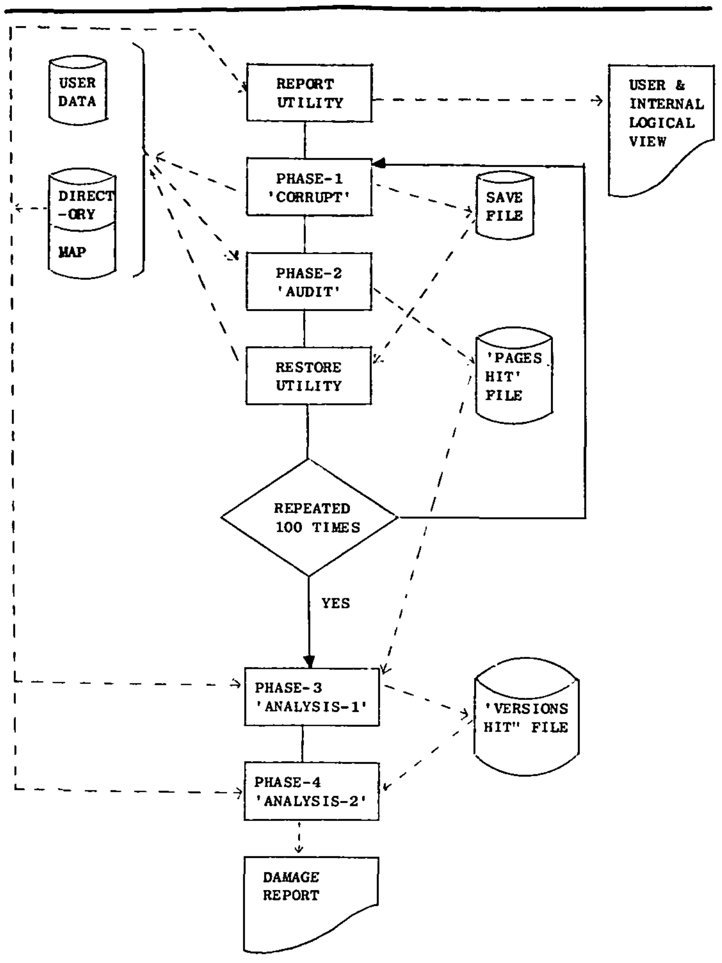

Fig. 1 Overall sequence of operations

position of the damage. AUDIT has knowledge of the design decisions made for page layout, location of control fields, etc. but it does not use any of the information held in the current directory or map (this could itself be corrupt). The auditing consists of trying to read each page in each of the three areas and performing consistency checks on successfully read pages. The nature of the consistency checks varies with the type of area and the type of record. Each page in the user data area has both user data records and system control data; the system data includes a page checksum, usage counts, the identity of the version of which this is a page and time and date last written. The checksum is recomputed and checked; the remaining fields are tested for radix and range and the free space in the page is checked to ensure that each unused word is set to 'null'. The format of pages in the system directory (folio and version headers, search addenda) is known exactly, thus every field is checked for radix and range and the relationship between fields is checked.

There are no checksums on system directory or system map pages. The system contains a mapping between logical pages and physical pages for every version. The map is followed (but without accessing the pages of user data) for each version, checking the radix and range of each map element and the particular format of the first and last elements. *All* the empty space in the map is checked for null element values. Whenever a page fails the appropriate set of checks its area name and page address are appended to a 'pages hit' file.

After the corrupt and audit phases have been run once the data base is returned to its original correct state using a utility program, RESTORE, which copies the contents of the 'save file' back to the appropriate part of the base. This whole process is repeated one hundred times creating a 'pages hit' file for subsequent analysis. The first two phases are separated from the analysis phases for operational convenience; the analysis phases do not require the user data area to be on-line and thus are more cheaply run apart from the corrupt and audit phases.

Analysis 1—physical to logical internal level

The third phase loads the first of the two analysis programs, this program processes the 'pages hit' file and produces a file of 'versions hit'. If the page hit is in the system directory area then the analysis performed depends on the type of record damaged. The type of the record is deduced from the inverse of the key-to-address transformation function used to locate records in this area. If the record damaged is a search addendum then no user information has been lost. Although the system has lost a set of search statistics this does not affect the validity of the version searched but means that the reorganisation of the version may be postponed since the statistical evidence supporting reorganisation must be collected again from the current time period. If the record position hit is a version header then the folio header is checked to see if this version position is in use; if the position is in use then the version identifier is written to the 'versions hit' file. If the record position hit is a folio header then the definitions of a complete set of versions have been lost and the system cannot continue without this folio header being reloaded. Fortunately folio headers occupy only a small part of the system directory area which itself occupies less than four per cent of the total space allocated to the data base.

When a user data area page is hit the system map is processed with each logical to physical map being followed until the value of the next physical address corresponds with the damaged page. The identifier of the version owning the map being processed is then written to the 'version hit' file. Suspect system maps are processed in a similar fashion. Each map is followed until the next map element would be on the damaged page. All the maps are processed since one page of the system map may have map elements from more than one map and thus more than one version may be inaccessible due to damage on a single page of the system map. The overall logic of this process is shown in **Fig. 2.**

Analysis 2—logical internal to user level

The fourth phase loads the second of the two analysis programs. This program reads the folio and version headers from the system directory and builds a table containing the description of the data base. A record from the 'versions hit' file is read and the part of the data base defined by the record as damaged is deleted from the data base description. The resulting data base description is restructured as a graph with each field represented as a node and each version represented as a set of arcs between the fields of that version. If the graph is connected then the user view of the data base is unchanged and the data in the damaged version can be accessed by some route using one or more other versions. In this case no action is required to return the data base to the correct logical state. If, upon analysis, the graphical representation of the data base description is more than one connected graph then to restore the data base to its correct form it is necessary to supply links between the graphs to create a single connected graph. The user is informed by the system of the fields within the groups making up each graph and of those fields with the fewest duplicate values to help him decide which links to provide. Thus the user may be able to provide a comparatively small portion of the data base and hence effect recovery without the high cost of a complete physical reload. This process is repeated for every record on the 'versions hit' file.

Experimental results

The CORRUPT and AUDIT programs ran one hundred times. This caused one hundred different parts of the data base to be

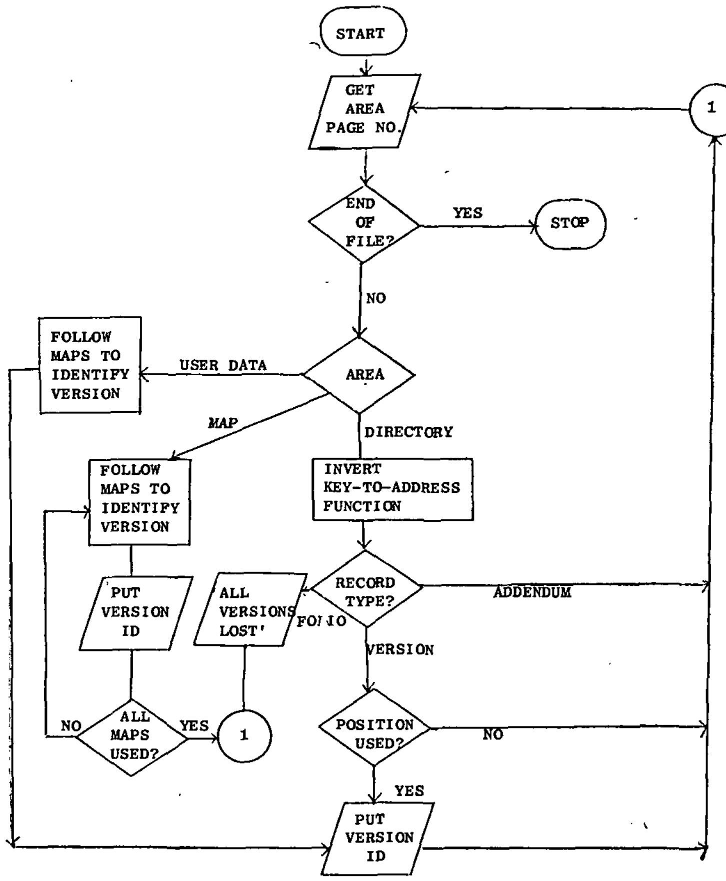

Fig. 2 Analysis 1—Physical to internal logical

Table 1 Experimental results from test data base

1. Page positions 'hit'	
System directory area	
Folio headers	0
Version headers	4
Search Addenda	1
System map area	
Starts of Maps	0
Map Element blocks	9
User data area	
Data pages	84
	98
Corrupt pages not found	2
Total	100
2. Versions lost due to 98 occurrences of corruption	11
User supplied linkages required to allow base to be used	0
Natural resilience allowed base to be used	11
Total	11

damaged. The audit program was able to detect 98 of the occurrences of damage; two occurrences were not found and no spurious occurrences were reported. The 98 occurrences were comprised of four version header positions, one search addendum position, nine pages of the system map area and 84 pages of the user data area. The first analysis program identified 11 versions which were damaged. The second analysis program found that in all 11 cases the version damaged was not essential and that the data base continued to be usable. This last result was due to the large amount of redundancy in the test data base (the data used is that example one in Dearnley, 1974b). The second analysis program was also tested on an artificially contrived set of damaged versions to ensure that the graph processing algorithms functioned correctly. These results are summarised in **Table 1**.

Cost comparison

If the number of disc accesses required to read or write the entire data base is n and the number required for the system directory and map together is m then the cost of the physical copy dump and restore method is $4(n + m)$, i.e. read the data base and write the dump followed by read the dump and load back to the data base. The cost of using the inherent resilience of the self organising system with the appropriate audit and analysis programs is $3m + \delta + \varepsilon$ where (*a*) if the data base is still usable δ is zero else δ is the cost of providing that set of links that join the usable section of the base and (*b*) ε is the cost of writing the 'pages hit' and 'versions hit' files. It is assumed that ε is insignificant. Thus the best case is $3m$ and for other cases the value of δ may vary considerably but it is reasonable to assume that it will always be less than copying the entire data base. Since $n \simeq 25m$ in the current system the value of δ can approach $4n$ before the physical copy system becomes economic.

Implementation and assessment

The results of the series of experiments suggest that it is worthwhile to implement a single program based on the audit and analysis programs used in the test. This program would be used to investigate suspected corruptions of the data base and to report any action which might be required to effect recovery. In addition to relying upon the inherent resilience of the self organising data management system it would be prudent to maintain a physical copy of the system directory and map. In the experiments performed the first phase never damaged a folio header, but if this occurred in normal operation access to all versions of the folio would be lost, hence some back-up facility is highly desirable. The data base used in the experiments contained considerable duplication which will not always be the case and a user of such a system may wish to ensure that complete recovery will always be possible. The user could keep a physical copy of the data base to either completely reload if the task of supplying missing links is too great for him or to select those parts containing the damaged versions by examining the system map. This combination of techniques would cost $5m + \delta + \varepsilon + 2n$; the addition $2n$ covers creating a physical copy of the user data area and the additional $2m$ is a copy of the system directory and map. Ignoring ε, as before, and assuming $n \simeq 25m$ then the combination of physical dumping and 'natural resilience' costs $55m + \delta$ against $104m$ for the normal physical dump and reload. Thus the 'best' saving is $49m$, when δ is equal to zero (as occurred in all the test cases) and the worst possible case is a loss of m when a complete reload is necessary and δ is equal to $2n$. Most cases would involve reloading a relationship of less than the entire data base and thus δ would usually be much less than $2n$ and a saving would accrue.

References

Browne, P. S. (1972). Computer Security—a Survey, *Data Base*, Vol. 4, No. 3.
Clifton, H. B. (1969). *Systems Analysis for Business Data Processing*, Business Books, London.
CODASYL Data Base Task Group Report, 1971.

CONWAY, R. W., MAXWELL, W. L., and MORGAN, H. W. (1972). On the implementation of Security Measures in Information Systems, *CACM*, Vol. 15, No. 4.
DATE, C. J., and CODD, E. F. (1974). The Relational and Network Approaches: Comparison of the Applications Programming Interface, Proc. 1974 ACM-SIGFIDET workshop on *Data Description, Access and Control.*
DEARNLEY, P. A. (1973). Low-cost File Security, *Management Informatics*, Vol. 2, No. 4.
DEARNLEY, P. A. (1974a). A Model of a Self-organising Data Management System, *The Computer Journal*, Vol 17, No. 1.
DEARNLEY, P. A. (1974b). The Operation of a Model Self-organising Data Management System, *The Computer Journal*, Vol. 17, No. 3.
FOSSUM, B. M. (1974). Data Base Integrity as provided for by a particular Data Base Management System, In *Data Base Management* edited by Klimbie, J. W. and Koffeman, K. L., North Holland.
PALMER, I. (1973). *Data Base Management*, Scicon.
STOCKER, P. M., and DEARNLEY, P. A. (1973). Self-organising Data Management Systems, *The Computer Journal*, Vol. 16, No. 2.
STOCKER, P. M., and DEARNLEY, P. A. (1974). A Self-organising Data Management System, in *Data Base Management*, edited by Klimbie, J. W. and Koffeman, K. L. North Holland.
STROSS, C. O. M. (1972). Operation of a Disc Data Base, *The Computer Journal*, Vol. 15, No. 2.
WILKES, M. V. (1972). On preserving the Integrity of Data Bases, *The Computer Journal*, Vol. 15, No. 3.

Book review

Microcomputers: Fundamentals and Applications, edited by G. Cain, 1975; 211 pages. (*Miniconsult*, £9·00)
Minicomputer Evaluation and Selection, edited by G. Cain, Y. Paker and P. Morse, 1975; 172 pages. (*Miniconsult*, £9·00)
Minicomputers: In Industrial Process Control, edited by G. Cain, Y. Paker and P. Morse, 1975; 172 pages. (*Miniconsult*, £9·00)

These three volumes form part of a series of (so far) six sets of course proceedings. They comprise the proceedings of three three-day courses given at the Polytechnic of Central London. It is one of the duties of polytechnics to provide education in technology by running both long and short courses. These volumes record how well short courses at a professional level are run by the Polytechnic of Central London. They are 'professional' in that they are aimed primarily at users and potential users of mini- and micro-computers. Though the courses are organised by academics, the authorship of the collected papers forms a nice mixture of academics and professionals each operating from their own standpoints.

The great advantage of organising courses as distinct from conferences, on technical subjects, is that the organisers can choose their authors and guide them as to what is wanted and how it should be presented. This allows the whole presentation to have continuity and completeness in a way that conferences often have not. It allows, too, the inclusion of papers and presentations of a basic and introductory nature. Thus the people attending the courses are all brought to within a common basis of knowledge and vocabulary so that they can benefit from the more advanced and specialised material which must, for many, be the main interest in the course. The organisers of these courses have had this point much in mind and so each of these volumes emerges as a text-book of a rather expensive, 'state-of-art' kind, even though it is a collection of papers by different authors. Thus, the books are truly useful to people who did not, or could not, attend the courses themselves. They are useful, too, in that they contain quite a lot of valuable basic and reference material such as glossaries, surveys and lists of participants which help the reader to gain an idea as to who, and what organisations, are interested in the subject material, in their capacities as makers, developers and users.

So much for the general review; the remainder of this review deals with the individual sets of proceedings in date order.

Minicomputers in Industrial Process Control, edited by Y. Paker, G. Cain and P. Morse; the course was held in March 1973.

This volume is aimed at process control engineers. Since the organisers were aware that some of the course participants were not over familiar with the entrails of digital computers, the proceedings start with a paper by Y. Paker on 'Basic principles'. This is a subject covered by many people many times before but rarely so well. This paper would provide a good introduction to any text on digital computers, whether mini or otherwise. It would be a good set of proceedings, indeed, if all were as good as Paker's, which they are not and so they suffer by comparison. Cain's paper which follows, entitled 'The Minicomputer as a control element' attempts to bridge the gap between what computers can do and what control engineers want to do, or should want to do. It is somewhat more flowery than Paker's paper and it loses by contrast also by leaning towards selling the mini as a device. A third introductory paper by D. J. Fraade, of Swiss Ciba-Geigy, is entitled 'Minicomputers in process industries'. It is somewhat replete with 'overviews'—which are 'in' things and some comparative cost studies. It loses in objectivity by being largely an exhortation by an enthusiast. Nonetheless, these three papers do make up a useful introduction to the papers which follow, which become increasingly detailed and instructive. The first of this group by B. West of UMIST is entitled 'Control of processes' and is considerably more down to earth and objective than its predecessor. This is followed by useful contributions on interfacing, software and system design as general topics.

The final group is made up of three interesting and well assembled applications lectures on 'Control in the oil and chemical industries' by J. D. F. Wilkie of GEC-Elliott, 'Continuous control in rolling processes' by M. Dean of British Steel Corporation and 'Sequence control in fine chemicals, plastic and dyestuff production' by P. Burton of Kent Automation. It would be difficult to make a significant improvement on these three applications as a cross section of the subject. In all, this is a useful book both for reading and reference.

Minicomputer Evaluation and Selection, edited by G. Cain, Y Paker and P. Morse, based on the course held in June 1973.

There are now many minicomputers available and many people who could afford to use them. A principal problem is to map the one

Continued on page 131