Methodology of computer systems design

S. J. Waters

London School of Economics, London WC2

This paper discusses the need for a systematic approach to computer systems design whereby networks of files and programs can be planned in a logical, orderly manner. Alternative methods are compared and developed into a design 'ladder'; each step in this ladder represents a design decision and an eventual climb of the ladder achieves a feasible computer system.

(Received June 1972)

This paper results from the CAM research project at the London School of Economics and Political Science, which is investigating a computer-aided methodology of developing computer-based, information processing systems; the project was initially financed by the Science Research Council and outlined by Waters (1972a).

At some stage in the development of an information processing system, the following design problem must be solved:

DESIGN	a computer system
SATISFYING	the defined objectives (and constraints)
GIVEN	a definition of information processing requirements and a definition of resources.

The *computer system* that is being designed is essentially a network of programs and files that is often represented by a 'program suite organisation flowchart' or 'run diagram'. The nodes of this network are the programs and the branches are the files of the computer system; the input/output interfaces between the computer system and its environment are also defined. **Fig. 1** illustrates a serially processed payroll system; much supporting documentation is necessary to detail this simple system: complex systems contain hundreds of programs and files.

The *defined objectives* to be satisfied include, at least, the dozen discussed by Waters (1972); these are efficiency, timeliness, accuracy, security, compatibility, implementability, maintainability, flexibility, robustness, portability, acceptability and economy. In practice, these objectives conflict and their relative weights vary between systems and designers; further, these relative weights are rarely quantified.

The *given definition of information processing requirements* includes the input/output messages, the database contents and the processes that transform input and database information into output and further database information. The *messages* are defined in terms of format, device/media, sequence, volume, frequency, response time, source/destination, etc; these messages connect the computer system to the organisation's human and automatic systems (e.g. new employee data, labour statistics) or to the organisation's environmental system (e.g. tax code assessments, tax returns). The *database* is defined in terms of information content, size, accessing activities, etc. but does not include partitioning into files which is one aspect of the computer systems design problem; Waters (1972) defines some of the common accessing activities (e.g. hit ratio, hit group, fan in/out ratio, volatility, point overflow). The *processes* are defined in terms of the conditions and transformations which process information.

The *given definition of resources* includes the computer hardware/software configuration which is either available or being proposed and the 'men, money, machines' and time that might also be available.

Thus, the computer systems design problem is clearly complex, particularly as a result of the large quantity of *given* data, the imprecise nature of an overall *objective function* and the intricate *design* process. In practice, alternative designs are also hypothesised by varying the *given* data and/or the *objective function* until a satisfactory solution is achieved.

The need for a systematic approach

Many existing computer systems do not meet the above dozen objectives, usually because:

1. There are severe communication problems between users, analysts, designers, programmers and operators.
2. Systems definition is often inadequate and inaccurate.
3. Insufficient time and effort is devoted to systems design.
4. Fallacious rules of thumb are widely used.
5. Designers rely on their own (limited) experiences; there is a tendency to make the current system 'look like' a previous one.
6. Systems are designed in insufficient detail.
7. A formal method has been lacking; most of the literature discusses techniques, not method.

This paper aims to overcome the latter three points by discussing a method that incorporates the major design decisions in a logical sequence. Instead of relying on (elusive) inspiration, the designer's thought processes are directed step-by-step through the design problem.

Langefors' method

Langefors (1966) published an early attempt at a design method within an overall approach to systems analysis that is basically sound; this work is still widely referred to and taken as a basis for much research in the field.

The method fails because the number of feasible groupings is vast, minimisation of transport volume is not the main (let alone single) objective and memory size constraints are relaxed by direct access storage devices. In particular, the detailed approach is obscure, unsystematic and far from complete; it is invalid for early, serial access computers as well as modern configurations.

Martin's method

The eminently practical Martin (1967) proposes an alternative method, but unfortunately, Martin's method is again incomplete; this paper extends his file/program approach to a set of sequential, detailed design decisions.

The proposed method

Computer systems design is but one stage in the continuing process of systems analysis, design, implementation and maintenance. Therefore this stage can be entered from any of the others and may itself enter any of the others; the total approach is usually iterative, not linear.

Fig. 2 illustrates an overall method of computer systems

(This paper was originally submitted in June 1972 as half of a longer paper; the second half will be published shortly.)

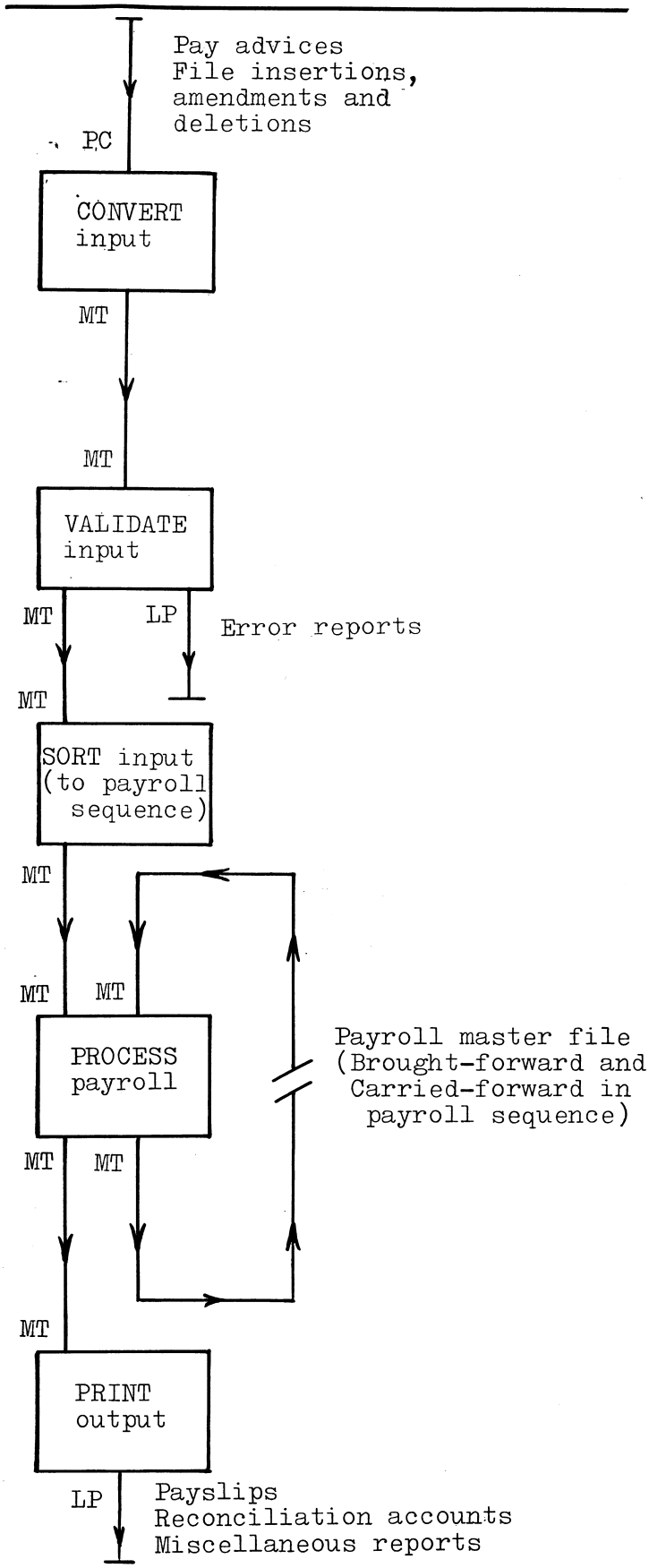

Fig. 1 Computer system for a serially processed payroll

design. The first two steps design the input/output files and programs. The next step designs logical master files by deciding information content, sequence, access method and format; logical master (e.g. process and update) programs are then designed by deciding their procedure content, processing modes, security and breakpoint facilities. Slave files (e.g. transaction, temporary and working) and slave programs (e.g. sort and

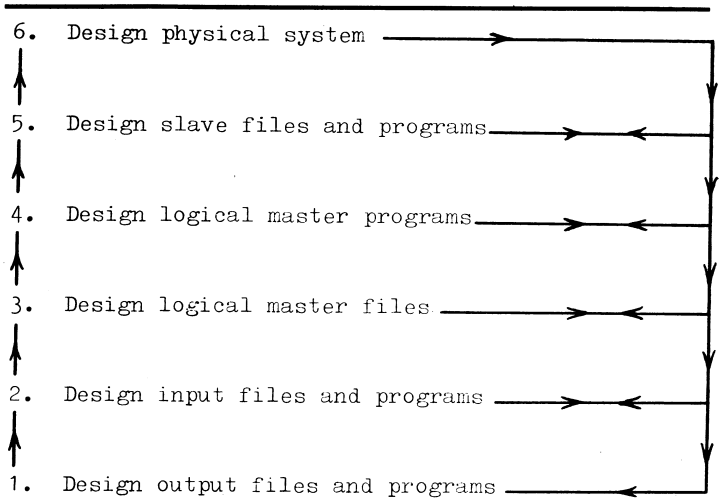

Fig. 2 Outline of a computer systems design method

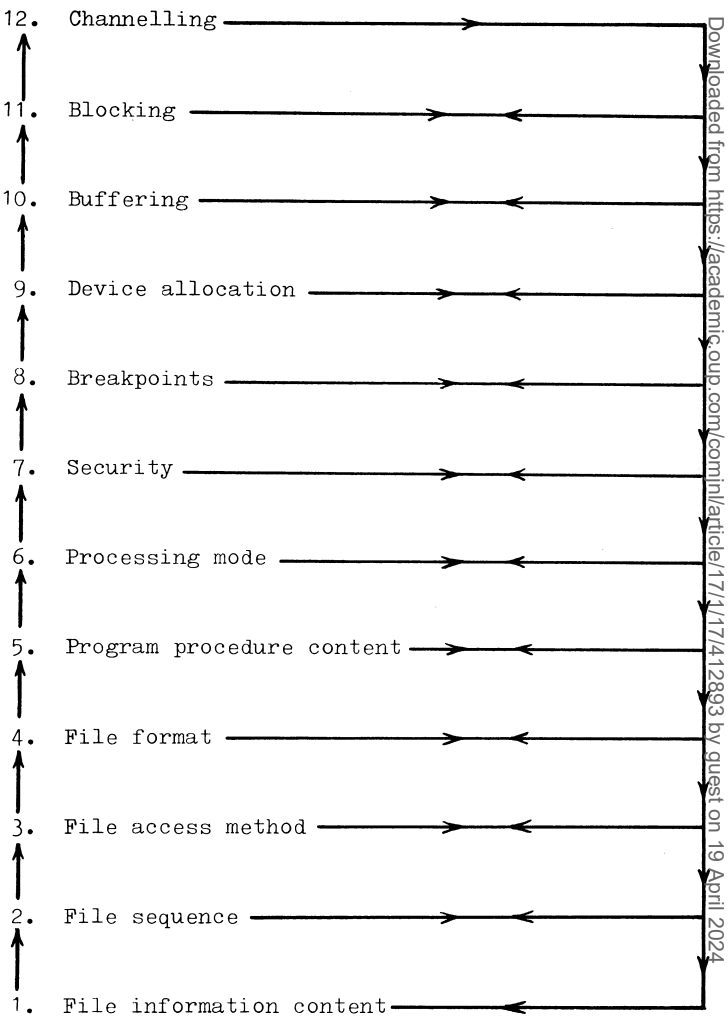

Fig. 3 A computer systems design 'ladder' of decision points

dump) are inserted to complete the logical system; finally, this is developed into a physical system by deciding device allocation, buffering, blocking and channelling arrangements.

Fig. 3 illustrates a proposed 'ladder' of computer systems design decisions; this 'ladder' is operated within the overall method of Fig. 2. Each step represents a decision point and the 'ladder' is sequentially climbed one step at a time; the current design can be estimated and assessed at each step with the possibilities of continuing the climb or slipping back to any previous step to take an alternative design approach. The first eight decisions are concerned with structuring a 'logical' system which satisfies the given information processing require-

FILE CHARACTERISTIC	FILE ACCESS METHOD		
	SERIAL SEARCH (Appendix 3; 1c)	ISAM (Appendix 3; 2b)	DIRECT ADDRESSING (Appendix 3; 3a(i)
Quick-response requirement	Doubtful (as time consuming)	√	√
Record keys sparsely allocated over key number range	√	√	Doubtful (as storage wasted)
Very large size	√	Doubtful (as necessary on-line storage unavailable)	
Volatile (high proportion of record insertions and/or deletions)	√	Doubtful (as overflowing can be significantly inefficient)	√
Point overflows (large groups of records are inserted with consecutive key numbers)	√		√

Fig. 4 Some guides to assist choice of key-retrieval, file access method for given file characteristics

ments; the first four structure logical files and the second four structure logical programs. The final four decisions are concerned with structuring a 'physical' system whereby the logical system is fitted to the given resources to define hardware utilisation. It is vital to recognise that a system is not designed until all decisions have been taken; each and every decision can have significant impact on the performance of the system. Appendices 1 to 12 discuss each decision and its alternatives with the factors that guide an efficient choice.

Some of the initial work contributing to this method was published by Waters (1970) and a detailed explanation of the approach with examples is awaiting publication in Waters (1973).

Conclusion

This paper has developed a manual method which is effectively a 'guided tour' through the highly technical 'jungle' of computer systems design. If nothing else, this method provides a sequential checklist of major design decisions with guides for each decision; as so often happens, investigating the application of computers improves insight into the equivalent manual operation. However, the CAM research indicates that computer software should yield substantial benefits by improving the quality of manual systems design without replacing the human designer.

The manual methodology has already been discussed and accepted by several industrial organisations and the author would be pleased to discuss it with any other interested parties.

Acknowledgements

The author wishes to acknowledge the assistance of his colleagues in the LSE Systems Research Group, particularly Mr. F. F. Land and Dr. A. H. Land of the Statistics, Mathematics, Computing and Operational Research Department.

Appendix 1 File information content

This decision is to group elementary information items into records and records into files (which constitute the database); it is assumed that conventional file organisation techniques will subsequently be used. The factors that guide the large number of alternative choices are:

1. The items and their identifiers or keys (e.g. gross pay to date identified by employee number, usage quantity identified by product number and resource number).
2. The relationships between identifiers (e.g. an employee works in one department which belongs to one factory).
3. The frequency of access to items and the identifiable subsystems that access them (e.g. product stock accessed continually for despatching, product price accessed daily for invoicing and product sales history accessed weekly for forecasting).
4. Identifier activities with respect to input/output messages and processes (e.g. hit ratio, hit group).

A record can be formed by grouping items as follows:

1. All items for common identifiers; for example, one record could contain all employee items (both payroll and personnel). Common identifiers can consist of several individual identifiers; for example, usage quantities would be identified by product and resource which could form product/resource records separate from basic product and basic resource records.
2. All items for common identifiers and a common accessing frequency and/or a common subsystem; for example, an employee payroll record and an employee personnel record. A sales control system might eventually justify a continually-accessed product stock record, a daily-accessed product pricing record and a weekly-accessed product sales history record.

In general, the first approach is preferable because consolidation of items into records reduces the number of files and simplifies their control (particularly with respect to identifier insertions and deletions). The second approach can subsequently be taken to avoid subsystems carrying overheads of irrelevant information.

A file can be formed by grouping records as follows:

1. All records for common identifiers; for example, a product file of all product records and a product/resource file of all usage information records.
2. All records for related identifiers; for example, a payroll file of employee, department and factory records. A production system might justify a single product file of all products with each record containing basic information followed by resource usage information.
3. All records for common and/or related identifiers constituting a hit group in a low activity situation; remaining records are grouped into a relatively inactive file. Sales ledger systems often justify a file of active customers and an archive file of inactive customers; high activity product records (e.g. catalogued products) could constitute a daily-accessed file and low-activity product records (e.g. special products) could constitute a weekly-accessed file, if acceptable.

Generally, the first two approaches would again achieve file consolidation whereas the last approach could avoid file transfer overheads.

Finally, this 'data processing' information that has been grouped into files may subsequently be enhanced by 'housekeeping' information that is a function of the design process; for example, labels, trailers, controls, etc.

Appendix 2 File sequence

This decision is primarily between a random or sequential file and, in the latter case, between a choice of sequences with respect to identifiers. In practice, random files are relatively rare; guides to the choice of sequence for a sequential file are:

1. The given information processing requirements might dictate that a particular procedure be obeyed in a specified sequence; organising the file in this sequence can avoid additional processing (particularly sorting). For example, product stocks might need to be allocated in customer priority sequence so organising the customer file in customer priority/customer code sequence could prove advantageous.
2. Large volumes of output messages are usually required in a specified sequence (if only to facilitate subsequent human referencing); again, organising the file in this sequence can avoid additional processing (particularly sorting and editing). For example, organising a payroll file in factory/department/employee sequence facilitates 'spinning-off' payslips in the required sequence without intermediate sorting and editing.
3. Conversely, a large volume of input messages could be submitted in a specified sequence, therefore organising the file in this same sequence could avoid initial sorting.
4. Processing can be simplified by organising the file in a suitable sequence; in particular, the 'pigeon-holing' technique for accumulating totals can be facilitated. For example, a payroll file organised in factory/department/employee sequence enables department and factory totals to be 'pigeon-holed' as a run progresses and only current department and factory totals need be accumulated.
5. The size of a file can be a function of its sequence, particularly due to the repetition of identifiers. For example, the simplest usage file might contain 18 characters of information for each of 10,000 product codes of ten characters and 1,000 resource codes of four characters; organising the file in product code/resource code sequence requires $10{,}000 \times (10 + 1{,}000\ (4 + 8)) = 120{,}100{,}000$ characters whereas resource code/product code sequence requires $1{,}000 \times (4 + 10{,}000\,(10 + 8)) = 180{,}004{,}000$ characters, giving a 50% increase.
6. File sequence could be chosen to enforce a hit group situation which might substantially reduce subsequent processing time. For example, a public utility (e.g. gas, electricity) system could organise the consumer file in the sequence of meter-reading round number; this enforces hit groups for the production of meter-readers' round lists and subsequent consumer bills and, to some extent, for payment updating.

A possible disadvantage in choosing a convenient file sequence as above is that a significant number of record key amendments might occur which require extra program runs and file passes to maintain the sequence of the file. Notice that quick-response systems usually restrict the choice to guides 5 and 6.

Appendix 3 File access method

This decision is to choose the addressing method whereby records of a file can be located and accessed.

Key retrieval is the usual technique whereby a key number is given and its record is accessed by one of the following methods:

1. Search: record keys are progressively examined until the matching record is found. Alternative searching methods are:

   (*a*) total search, whereby all records are examined from the first; the file can be sequential or random;

   (*b*) partitioned search, whereby all records are examined from the appropriate partition point (found by an index or algorithm); the records within a partition can be sequential or random;

   (*c*) serial search, whereby all records are examined from the previously matched record; the accesses are made in the same physical sequence as the file;

   (*d*) logarithmic search (or binary chop or repeated dichotomy), whereby a sequential file is progressively bisected.
2. Index: record key is translated to record position by searching index(es) instead of the file; alternative indexing methods are:

   (*a*) full index, whereby a single index defines the record key/position for every record in the file; the file can be sequential or random;

   (*b*) hierarchical (or partial) index, whereby level n index ($n = 1, 2, \ldots N$, usually $N = 2$ or 3) defines the highest key number in each of the level $n + 1$ indexes until level N index defines record position: ISAM uses this approach to reduce the number of seeks to one, for most accesses.
3. Algorithm: record key is translated to record position by arithmetic and/or logical transformation; some alternative algorithmic methods are:

   (*a*) simple algorithms, whereby a basically sequential file is generated without synonyms; examples are:

   (i) direct addressing (or self-indexing) where

   $$\text{Position} = \text{Key No.}$$

   (ii) 'degapping' which is a refinement of direct addressing whereby large blocks of consecutive, unallocated key numbers (i.e. 'gaps') are removed before applying

   $$\text{Position} = \text{Reduced Key No.}$$

   (iii) scaling, which assumes even distribution of allocated key numbers over the key number range with

   $$\text{Position} = \frac{\text{Key No.} \times \text{No. of Records in File}}{\text{No. of Key Nos. in Range}}$$

   (*b*) complex algorithms, whereby a random file is generated with synonyms; examples are prime division, digit selection, folding, truncation, radix transformation, squaring, etc.

Fig. 4 illustrates some guides for the common key-retrieval file access methods; Waters (1972) indicates the fallacy of basing access method on hit ratio so this is omitted from the

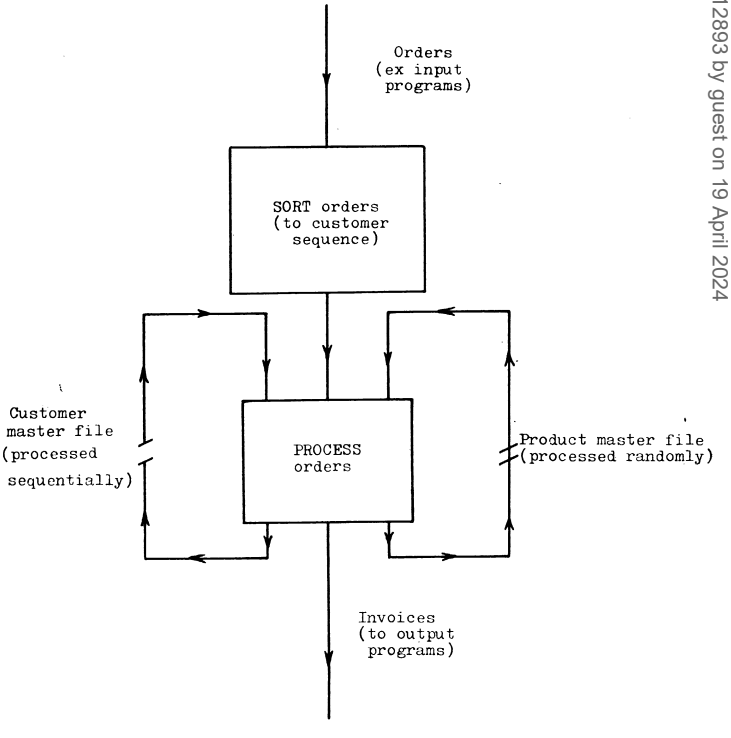

Fig. 5 'One-shot' processing

guides. The available secondary storage devices can restrict choice of file access method; a limitation of only serial access devices (e.g. magnetic tapes) implies serial searching; direct access devices with zero seek time (e.g. fixed head drums and discs) may prefer full or hierarchical indexing to the usual ISAM approach.

If there is an additional accessing requirement for content retrieval, whereby records satisfying specific conditions are to be retrieved, then the above file access methods require a total search of the file to achieve this. If this is unacceptable, then (partially) inverted file organisation allocates an index to each condition which defines each record key satisfying that condition; thus, provided the conditions are predictable, lengthy file searches are reduced to index searches. Alternatively, multilist file organisation allocates a list to each such condition to reduce file searches.

Appendix 4 File format

This decision is to define the format of the file, its records and their items.

The format of an item of information can be fixed or variable length; fixed length format further permits decimal or binary representation for numeric items, otherwise character representation is usually chosen; for example:

1. Fixed length binary format (e.g. word format) is sometimes used for numeric items to utilise faster arithmetic operations without suffering input and/or output conversion.
2. Fixed length binary format is sometimes used for limited value items in a large file to reduce file size by 'packing' (e.g. an indicator can be packed into 1 bit).
3. Character format is probably used more widely; although variable length format can significantly reduce file size, fixed length format is often chosen to meet software constraints.

The format of a record can be fixed or variable length. Again, variable length format can significantly reduce file size where maximum record length is substantially greater than normal record length; however, fixed length format is often chosen to meet software constraints. In extreme cases, a variable length record can be segmented into a variable number of fixed length records to meet such constraints.

The format of a file can be fixed or variable length. Usually, files contain variable numbers of records and therefore have variable length; however, fixed length files occur when:

1. Access method (e.g. direct addressing) dictates this requirement.
2. File size is significantly reduced by identifying each record by its position (instead of its key number) when key numbers are densely allocated; for example, a product file of 9,000 active records of 10 data characters with a four-character product key would require 9,000 (10 + 4) = 126,000 characters if only active records are held; the equivalent fixed length file would contain 10,000 × 10 = 100,000 characters, where the saving is due to eliminating key numbers.

Appendix 5 Program procedure content

The preceding four decisions have yielded a set of (logical) master files with information content, sequence, access method and format defined; this decision is to define a network of (logical) programs that satisfy the information processing requirements of the system (with little regard to computer configuration). Initially, feasible master programs are defined which process the master files; then slave programs (e.g. spool, data edit, sort, print, etc.) and slave files (e.g. accepted data file, print file, working files, etc.) are inserted to complete this logical system.

Initially, processes are grouped against master files with respect to common elementary information items; thus, all processes that refer to items of a master file are collected into a process program. This approach achieves program consolidation and reduces the number of file passes; the results depend on the number of master files and their degree of interaction with respect to processes as follows:

1. In many cases, the master files do not interact so that each master file generates a single process program. A common solution, in practice, is illustrated by the Fig. 1 process program; if several such process programs are generated then they can be further consolidated by 'piggy-backing' the master files. Occasionally, a process requires the master file be passed more than once; for instance, the record key amendments requirement that was mentioned above.
2. In some cases, the master files do interact because processes refer to several master files alternately; for example, the process that values customer orders for finished products

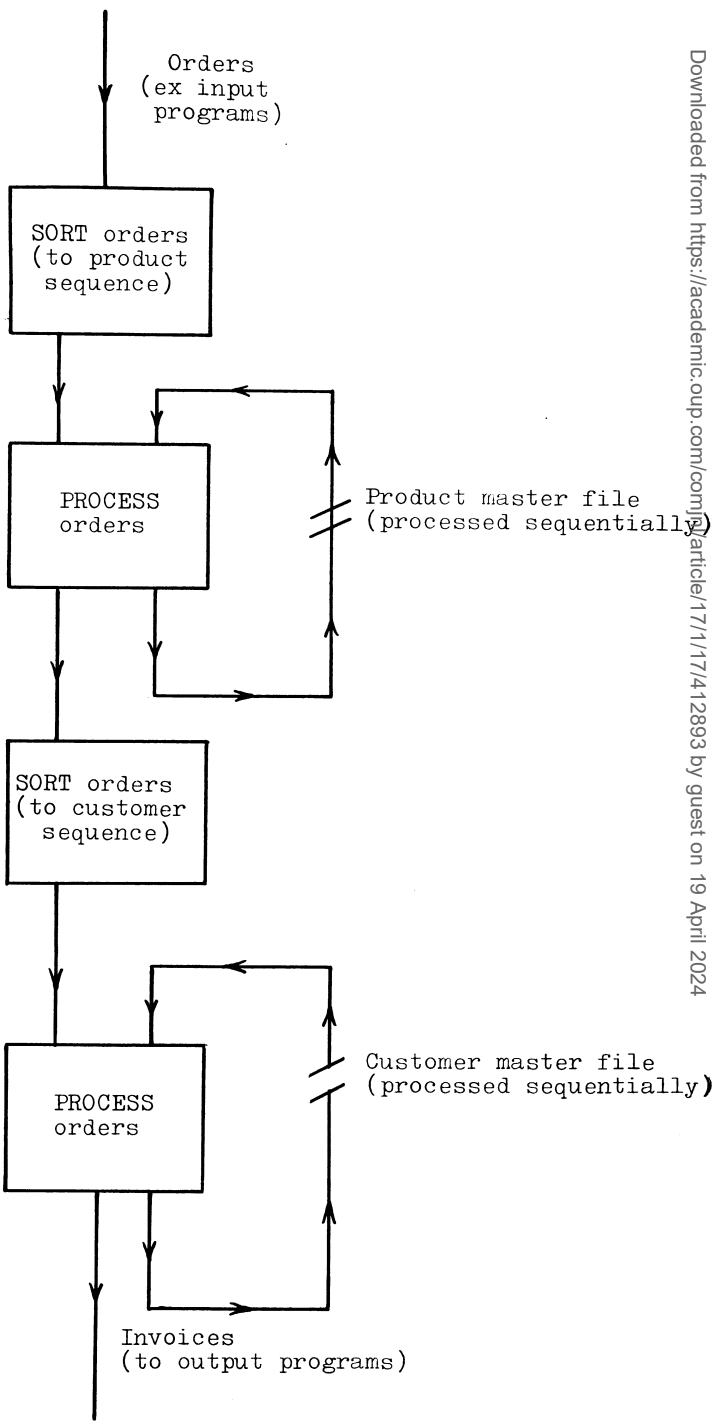

Fig. 6 'Sort-sequential' processing

FILE ORGANISATION METHOD		FILE PROCESSING MODE				
FILE ACCESS METHOD	FILE SEQUENCE	SERIAL	SEQUENTIAL	SKIP-SERIAL	SKIP-SEQUENTIAL	RANDOM
Search	Sequential	√	√	(Binary chop)	(Binary chop)	(Binary chop)
Search	Random	√	×	×	×	×
Index	Sequential	√	√	√	√	√
Index	Random	√	√	√	√	√
Algorithm	Sequential	√	√	√	√	√
Algorithm	Random	√	×	√	√	√

Fig. 7 Analysis of file processing mode against file organisation method
Total file transfer time = Maximum [(CPU Time) + Total of Column 1 File Transfer Times,
Column 2 Channel Transfer Times,
Column 3 File Transfer Times]

would first access the product file to establish prices and then access the customer file to establish trading terms (e.g. discounts). An extreme approach is the 'one-shot' solution illustrated by **Fig. 5,** as used in quick-response systems, whereby the process program has on-line access to all master files. Another extreme approach is the 'sort-sequential' solution illustrated by **Fig. 6** whereby the interactive processes are disected against the master files; care must be taken to readjust the initial updating of master files by a transaction if that transaction is subsequently rejected against the final master files. Between these two extremes, there are many permutations to the approaches depending on the number of interactive master files, for example, four such master files can yield some 20 variations.

These monolithic process programs can subsequently be split to yield alternative designs; common divisions are by process frequencies and/or message types.

Appendix 6 Processing mode

This decision is to define the processing mode of each file as accessed by each program. Generally, there are five possibilities:

1. Serial, by which the entire file is processed in its physical sequence. A file without index or algorithm facility would usually be serially processed; an indexed or algorithmic file can be serially processed to minimise seek time. This processing mode must be chosen if only serial access devices are available.
2. Sequential, by which the entire file is processed in its logical (key) sequence. This is equivalent to serial processing for a sequential file without an index facility. Indexed sequential and indexed random files can also be sequentially processed; the remaining random files can only be sequentially processed by accessing every possible key value (whether allocated or not) in consecutive sequence.
3. Skip-serial (or selective-serial), by which only active (blocks of) records are processed in physical sequence. Indexed sequential and algorithmic sequential files can be skip-serially processed by pre-sorting accesses to key sequence. Indexed random and algorithmic random files can be skip-serially processed by pre-sorting accesses to record address sequence; the record address is established before sorting by driving the key number through the index or algorithm. The remaining files cannot be skip-serially processed (other than possibly binary chopping a sequential file after pre-sorting accesses to key sequence).
4. Skip-sequential (or selective-sequential), by which only active (blocks of) records are processed in logical sequence. Indexed and algorithmic files can be skip-sequentially processed by pre-sorting accesses to key sequence; the remaining files cannot (other than possibly binary chopping a sequential file after pre-sorting accesses to key sequence).
5. Random, by which only active (blocks of) records are processed in neither physical nor logical sequence. Indexed and algorithmic files can be randomly processed but the remaining files cannot (other than possibly binary chopping a sequential file).

Fig. 7 summarises processing modes against file organisation methods.

Guides to the choice of processing mode, in addition to the above restrictions, are:

1. Random processing usually requires the entire file be on-line which might exceed direct access device capacity; this is occasionally overcome by randomly processing the file in sections that are 'storable'.
2. Random processing often incurs time-consuming seeks unless the file is small or the seek area is reduced by a hit group situation or the available direct access devices have zero seek time.

FILE CHARACTERISTICS (DEVICES, BUFFERS, CHANNELS)	SIMULTANEOUS OPERATIONS		
	PARTIAL (WITH MULTIPLE-BUFFERED FILES ON OTHER CHANNELS ONLY)	PARTIAL (WITH FILES ON OTHER CHANNELS ONLY)	TOTAL (IRRESPECTIVE OF CHANNELLING)
1. Single-buffered files	√		
2. Multiple-buffered, slow-speed device files (one file per device)			√
3. Multiple-buffered, magnetic device files—sharing a device		√	
4. Multiple-buffered, magnetic device files—separate devices but sharing a channel		√	
5. Multiple-buffered, magnetic device files—separate channels			√

Fig. 8 Aggregation of file transfer times to total file transfer time for a program

3. Random processing is usually necessary in quick-response systems unless file access queues are pre-sorted.
4. If fan in/out ratios are high (i.e. each hit record is accessed many times) then random processing causes multiple accesses to each hit block of records; other processing modes cause a single access only.
5. Updating by overlay (when skip-serially, skip-sequentially or randomly processing) subsequently requires complicated security and breakpoint procedures.
6. Updating by overlay can reduce secondary storage requirements because the brought-forward file is not updated to a physically separate carried forward file.
7. One-shot processing implies randomly processing some, possibly all, files accessed by the program.
8. Slave files are usually processed serially.

Waters (1972) indicates the dangers of basing processing mode on hit ratio but very low activity can favour skip-serial, skip-sequential or random processing; in particular, it is often more efficient to process queues of quick-response messages skip-sequentially by scanning than randomly (even if the 'minimum seek time' method is used).

Appendix 7 Security

This decision is to ensure that each file can be regenerated should it be lost, 'crashed' or corrupted. Generally, there are three possibilities (and combinations of the three):

1. Generation, where recent successive versions of the file are retained together with their updating messages; the 'grandfather system' commonly retains the current 'son file', its 'father file' and its 'grandfather file' and the 'son' file overwrites its 'great-grandfather' file. This method is used for serially and sequentially processed files where the brought-forward version is updated to a physically separate carried-forward version.
2. Dumping, where a file is periodically dumped to provide back-up copies; this method is used for skip-serially, skip-sequentially and randomly processed files which are updated by overlaying brought-forward records by carried-forward versions. If a file is to be recreated from its previously dumped version, then there are two further choices:
   (*a*) either the dumped file is updated by appropriate messages (which must also be retained) in the normal manner;
   (*b*) or the dumped file is updated by replacing original records by their new versions (which requires that the new version of a record be dumped whenever it is updated).
   The second method requires a separate secondary storage device from the file itself but can simplify the recreation procedures and save computer rerun time.
3. Duplication, where identical versions of the file are continually updated, whatever the processing mode. An advantage is that little computer time is lost in switching to the second version of the file if the first version fails; however, additional secondary storage devices can be required and computer time can be increased if simultaneity is not subsequently possible.

Thus, the method of master file security depends on file processing mode, acceptable recreation time and availability and reliability of computer hardware/software; file duplication is a simple but underused technique, particularly in serial processing systems. Usually, temporary files are recreated by rerunning the programs that generated them; again, file duplication is an underused but effective technique.

Appendix 8 Breakpoints

This decision is to choose breakpoint (or checkpoint or restart) procedures so that a program/file breakdown can be speedily corrected. In practice, batch-processing breakpoints occur at 15 or 20 minute intervals so that the program can be restarted from the last breakpoint instead of right from the beginning.

Appropriate areas (e.g. run totals), if not all, of primary storage are dumped at each breakpoint so that they can subsequently be reconstituted. The actual breakpoint is identified for all files by inserting special breakpoint records or by dumping the last record key before the breakpoint.

When a restart is necessary, the contents of primary storage are reconstituted and the files are re-aligned at the breakpoint as follows:

1. Serially and sequentially processed files are merely positioned at the first record after the breakpoint.
2. Overlayed files are sometimes dumped at each breakpoint, particularly if they are relatively small; these files are positioned by setting up the dumped version. Notice that random processing requires the entire file be dumped at each breakpoint but skip-serial processing only requires the current 'interval' of the file be dumped.
3. Otherwise overlayed files must be downdated from the breakdown back to the breakpoint; this requires the old versions of updated records to be dumped so that the file can subsequently be downdated. The combination of file security and breakpoint procedures often requires 'incremental dumping' whereby both old and new versions of updated records are dumped to a physically separate file; thus, the master file can be updated from a dumped version or downdated from the current version.

Appendix 9 Device allocation

The preceding eight decisions have yielded a logical computer system of programs and files with little regard to computer configuration; this must now be developed into a physical computer system by deciding hardware utilisation. This initial decision is to assign the following files to computer storage devices:

1. Master files, which are referenced or updated by each run of the system.
2. Slave files, which transfer information between programs within each run of the system.
3. Dump files, which support the security and breakpoint procedures.
4. Programs, which can be regarded as files having the following properties:
   (*a*) each program is a relatively small file; each record is a (housekeeping or information processing) procedure identified by a procedure name; each item is an operation;
   (*b*) the system only references such files; at present, they are updated by standard software systems of preprocessors, compilers and assemblers;
   (*c*) at present, each record is accessed by the operating system; usually, this accessing is random using an indexed or partitioned file.

Future developments might allow programs to be organised, updated and accessed by the information processing system to avoid the restrictions of current software systems; programs and files could then be combined and revised as convenient to the information processing system.

These files are allocated to devices to meet design objectives (often reduction of expensive computer time) subject to the following constraints:

1. The device must support the file access method and processing mode.
2. File security constraints must be met; thus, dumped and duplicate versions of master files must be assigned to separate devices from the master files as must those slave files that support the security system.
3. Breakpoint constraints must be met; thus, incrementally

dumped files must be assigned to separate devices from the master files.
4. An updated file must be output to the same type of device that subsequently inputs it; however, this constraint is sometimes relaxed by job control languages.
5. The number of devices and their capacities must not be exceeded.

The underlying implication of this generalisation is that interactive records from various files must be input to primary storage together; a program procedure and the information records it accesses must be simultaneously accessible by the logic and arithmetic circuitry of the central processing unit.

This device allocation problem often has a large number of solutions which can be reduced by the following guides:

1. Small, frequently accessed files are often allocated to primary storage and initially loaded from secondary storage devices; for example, common program procedures and reference tables.
2. Slave files supporting the security and breakpoint procedures are often allocated to magnetic tape devices.
3. Large, serially processed files are usually allocated to magnetic tape devices; in particular, large files are allocated to faster tape handlers if a variety is available.
4. Two devices are often allocated to 'multi-device' files that are processed serially or skip-serially to permit 'ping-ponging'; for example, one tape of a multi-reel file can be on-line while the previous tape is rewound and the following tape is loaded.
5. Operating systems often allow a file to be allocated to non-contiguous areas of direct access devices, where necessary.
6. The 'split-cylinder' technique is sometimes used to reduce time-consuming seeks on direct access devices where several files are serially processed together on the same device; for example, a brought-forward master file might be allocated to tracks 0 to 7 of each cylinder on a magnetic disc pack and its slave transactions file might be allocated to tracks 8 and 9.

   Otherwise, the most frequently accessed files are allocated to the middle cylinders so that the least frequently accessed files are allocated to the outer cylinders.

Waters (1972) indicates the fallacy of basing device allocation on file hit ratio so this is omitted from the guides.

Appendix 10 Buffering

This decision is to allocate primary storage buffers to secondary storage files to achieve simultaneity between file transfers and central processing unit operations. Slow-speed devices are usually double-buffered but fixed-cycle devices (e.g. some card readers) are often multiple-buffered to avoid missing cycles.

Skip-serially, skip-sequentially and randomly-processed files are usually single-buffered as the position of the next required record is not known in advance and therefore cannot be precalled while the current record is being processed; thus, there is no simultaneity between such files, other than some pre-seeking. If, however, the next record is known in advance then it can be precalled into a second buffer to achieve simultaneity; this assumes the file is referenced only, otherwise a return seek would be necessary to overlay the current record when it is to be updated after processing.

Serially and sequentially processed files are usually double-buffered to achieve simultaneity. Occasionally, small files are single-buffered to release primary storage and large, low-activity files are multiple-buffered to smooth any imbalance between file transfer and heavy record processing.

Appendix 11 Blocking

This decision is to choose block size so that secondary storage device files have reduced transfer time overheads and/or reduced file size; notice that all such devices are involved and not just magnetic tapes.

Often, installation standards or restrictive software constraints dictate standard block sizes (e.g. 1,000 or 2,000 characters); given an unrestricted choice, then block sizes are chosen as follows:

1. Waters (1971) developed a simple algorithm for serially and sequentially processed files, which apportions available primary storage to such secondary storage files by minimising computer run time; the algorithm caters for variable numbers of buffers, variable devices, multi-device files, etc. and achieves significant savings over traditional approaches.
2. Direct access device files are often blocked so that file size is reduced; the aim is to minimise block overheads and track wastage.
3. Skip-serially, skip-sequentially and randomly-processed files are often unblocked to avoid overhead transfer of spurious records; however, such files are serially or sequentially processed on occasion (if only for dumping or reorganisation purposes) and this strongly favours a large block size. This imbalance can be resolved by choosing an intermediate block size (which reflects the relative frequencies of the various processing modes).

Appendix 12 Channelling

This decision is to allocate the secondary storage devices to channels to maximise their simultaneity of transfer. Slow-speed (i.e. non-magnetic) devices are usually permanently assigned to multiplexor channels which (together with multiple-buffering) ensures simultaneity among them and the central processing unit.

High-speed (i.e. magnetic) devices are usually linked to selector channels so that only one device can use the channel at any point of time. Single-buffered files (e.g. those processed skip-serially, skip-sequentially and randomly) cannot transfer simultaneously to each other but multiple-buffered files can, provided they are assigned to separate channels. Some computers contain floating channels which achieve this simultaneity automatically by transferring information along any channel that is currently free; however, most computers contain fixed channels and devices are manually allocated to these channels to reduce computer time by balancing file transfer time over channels; a constraint may be to allocate duplicate files to different channels (to counteract channel corruption). **Fig. 8** illustrates the simultaneity achieved by buffering and (fixed) channelling.

References

LANGEFORS, B. (1966). *Theoretical Analysis of Information Systems*; Studentlitteratur, Sweden.
MARTIN, J. (1967). *Design of Real-Time Computer Systems*; Prentice-Hall, New Jersey.
WATERS, S. J. (1970). Physical Data Structures. Paper 6 of Proceedings of BCS Conference on *Data Organisation.*
WATERS, S. J. (1971). Blocking Sequentially Processed Magnetic Files; *The Computer Journal*, Vol. 14, pp. 109-112.
WATERS, S. J. (1972). File Design Fallacies; *The Computer Journal*, Vol. 15, pp. 1-4.
WATERS, S. J. (1972a). A Survey of CAM and its Publications. Final Paper of Proceedings of NCC Conference on *Approaches to Systems Design.*
WATERS, S. J. (1973). *Introduction to Computer Systems Design*, National Computing Centre (awaiting publication).