Management Handbook of Computer Usage

Management Handbook of Computer Usage

R K Sachdeva

British Library Cataloguing in Publication Data

Sachdeva, Rajinder
Management handbook of computer usage.
1. Computer systems. Management
I. Title
004'.068

ISBN 0-85012-769-6

Published for NCC Publications by NCC Blackwell Limited.

Editorial Office, The National Computing Centre Limited, Oxford Road, Manchester M1 7ED, England.

NCC Blackwell Limited, 108 Cowley Road, Oxford OX4 1JF, England.

Typeset in Times by Bookworm Typesetting, Manchester; and printed by Hobbs the Printers of Southampton.

ISBN 0-85012-769-6

Contents

1 Evolution of Computer Systems

EARLY COMPUTING DEVICES

The development of computers has followed a long evolutionary path. Man's interest in mechanical computational devices is age old: primitive mechanical computational devices can be traced to Ancient China and Ancient Greece. One of these, the abacus, is still in use in some parts of the world. It consists of a frame with beads sliding on wires to represent and manipulate numeric values.

Logarithm tables were first published by the Scot John Napier in 1614. Every positive number can be associated with a logarithm, with multiplication and division carried out through the simple operations of adding and subtracting logarithms. The slide-rule, based on the concept of the logarithm, was developed by William Oughtred in 1622. This mechanical device allows multiplication and division to be performed by sliding a moving rule against fixed calibrations. The slide-rule is an early example of an *analogue* device which employs continuous physical quantities, like length or voltage, rather than discrete numbers as in *digital* calculators and computers.

The arithmetic engine (the 'Pascaline'), produced by Blaise Pascal in 1642, was the first real calculating machine. Digits from 0 to 9 were arranged on wheels. Turning one wheel a full revolution caused its neighbour to be advanced one notch. The arithmetic engine worked on a similar principle to mechanical milometers or the taxi meter. Later, in 1694, a more sophisticated

mechanical calculator was invented by Wilhelm Gottfried von Leibniz. But it was not until the nineteenth century that mechanical computation took a major step forward. The Difference Engine, built by Charles Babbage, was unveiled to the Royal Astronomical Society in 1821. It calculated the differences between numbers to compute and check mathematical tables.

Babbage then began development of a new device, the Analytical Engine, that was intended to function as a program-controlled machine. This design provided 'architectural' concepts that were to prove important for modern electronic digital computers. The design for the Analytical Engine contained five elements: input, store, arithmetic unit, control unit and output. These, in one form or another, are intrinsic to all computer systems. Input was accomplished by means of punched cards, a facility originally developed by Joseph Maria Jacquard in France to control looms automatically.

The Unit Record System, also using punched cards, was developed by Hermann Hollerith in the 1880s. He won a competition to find an efficient way of analysing the 1890 American census. The 80-column card designed by Hollerith was still the main input medium in most electronic computers until the 1970s. The presence or absence of a hole in particular positions on a card would be used to indicate the presence or absence of particular characteristics of the individuals in the census. In order to read the cards, rods were passed through them. The rods then made contact with a bowl of mercury to form an electrical contact, causing a counter to advance by one. The Unit Record System consisted of a 'sorter' for arranging the punched cards in a desired sequence, a 'collator' to selectively merge the cards and a 'tabulator' to tabulate the contents of the cards.

This device was the first electromechanical computing system. Use was made of hard-wired programs to control the operations of the individual machines, and control panels could be wired with different programs to achieve different results from the same machines. Dr Hollerith's work eventually gave birth to International Business Machines (IBM) in the US and to International Computers Limited (ICL) in the UK.

DEVELOPMENTS IN COMPUTER HARDWARE

The concept of *stored program control* was further developed by John von Neumann in 1945 and this helped to lay the foundation for modern electronic computers. He built a machine – EDVAC, the Electronic Discrete Variable Automatic Computer – based on this principle, in 1951. An earlier device – ENIAC, the Electronic Numerical Integrator and Calculator – developed in 1946, is now seen as one of the first general-purpose computers. It was a gigantic machine with eighteen thousand valves. It consumed many kilowatts of power, had to be water cooled and had a very high failure rate compared to modern computers.

The earliest computers, built from electronic valves, are now depicted as *first-generation*. The introduction of transistors, in the late-1950s, resulted in *second-generation* computers. These were typified by the ICT (a forerunner of ICL) 1301 and the IBM 1401. *Third-generation* computers are based on integrated circuits where large numbers of transistors and other electronic components are carried on semiconductor *chips*. The ICL 1900 series, System 4 and the IBM 360, introduced in 1964, are examples. The development of large scale integration (LSI) and very large scale integration (VLSI) led to *fourth-generation* computers. (It was semiconductor technology which facilitated the emergence of microcomputers in the early-1970s). *Fifth-generation* computers, currently being developed, are intended to embody artificial intelligence (AI) and other features. They will, for example, be able to imitate common human functions like vision, natural language comprehension and the use of specialised knowledge.

The internal computer storage ('memory') was, in the 1940s, built from vacuum tubes. These were relatively large (a few inches in length), and each was able to hold only one binary digit, called a *bit* (a 0 or 1). The storage capacity of such first-generation systems was tiny by present standards. The most popular computer in the mid-1950s (the IBM 650) used a rotating drum coated with a magnetisable material as the primary storage medium. Between 1960 and 1975, however, the dominant computer storage design used tiny rings or 'cores' of magnetisable material in the primary storage section. Current flowing in one direction represented a 0 while flow in the opposite direction

represented a 1. Since the core permanently retained its magnetic state in the absence of the current, it was a non-volatile storage medium. Core storage was popular for 15 years because it was safe, durable and reasonably fast. However, the new storage devices that appeared in the 1970s offered even faster performance at a lower cost and so the popularity of cores quickly faded.

Virtually all today's computers use semiconductor elements in their primary storage sections. Semiconductor storage elements are tiny integrated circuits (ICs). Both the storage cell circuits and the support circuitry needed for data writing and reading are packaged on chips of silicon. The chips used for the primary storage section usually employ metal-oxide semiconductor (MOS) technology. In third-generation computers, the level of integration was of the order of 200 transistor components on one IC. The LS chips had a level of integration of 10,000 components. Present-day VLSI chips integrate one million or more components and can store that many bits of data. These chips are available for a few dollars each. This represents a reduction, from second-generation times, in the cost-to-performance factor of more than one thousand.

In the earliest electronic computers the feeding of input data to, and output from, the computer was via the medium of punched paper cards or punched paper tape. Such facilities were very slow compared to present standards. Second-generation computers used magnetic tapes which could store 800 characters of data per inch of tape length. Hard magnetic disks could store up to two million characters of data.

Miniaturisation of the central processing unit of the computer was also reflected in the input and output devices. Magnetic tape gave way to the cassette tape and the hard magnetic disk to the floppy in the smaller mini and microcomputers. Today 3½ inch-diameter floppy disks can store more than a million characters of data, and magnetic tapes with the capacity to store more than 6000 characters per inch are quite common. Hard disks used on larger computers can store more than 500 million characters. Similarly, rates of data transfer between the input/output devices and the central processing unit have increased from a few thousand characters per second in the second generation to more

than a million characters per second in today's systems.

Another popular modern input/output device is a visual display unit (VDU) or visual display terminal (VDT). The terminal has given rise to interactive computing – with on-line access to computer data and to the computing power needed to manipulate the data. Specialised input devices – such as optical readers in the point-of-sale terminal – have made it possible to directly feed data from the point of activity, instead of transcribing it onto paper cards or magnetic tape.

DEVELOPMENTS IN COMPUTER SOFTWARE

Earlier computers were sold as bare machines consisting of hardware alone. While the computer devices, such as the central processing unit and the input/output devices, form the hardware of the computer, processing procedures are necessary to solve user problems on the computer. These procedures (*programs*) are specified in special computer languages, and constitute the computer software. Modern computers are sold along with a wide variety of software designed to aid the computer user. Over the years, computers have become increasingly 'user-friendly'.

Communication with earlier computers was by means of highly complicated machine languages. Each instruction in a machine language consisted of a long string of binary digits (0 and 1). Each machine instruction performed a very elementary operation – such as moving one digit of data from one place in the memory to another or adding or subtracting two digits. The user operations had to be broken down to series of such elementary operations which could be directly executed by the machine. So programs for simple jobs consisted of a large number of instructions, each instruction in turn consisting of a long string of 0s and 1s. Each computer model had its unique machine language, and programming a different model of computer meant learning a different machine language. The machine languages are referred to as first-generation languages.

Second-generation languages replaced the binary codes with 'mnemonic codes' and 'symbolic addresses' consisting of alphabets and numerals. However, there was still a one-to-one correspondence between the program instructions and the

corresponding machine-language instructions. So the program still consisted of a large number of instructions performing elementary steps. Such languages are referred to as *low-level languages* and are also unique for each computer model. The program in such a language has to be first translated into the machine language before it can be executed; the translation is carried out by the computer itself using the software supplied. Examples of second-generation languages are AUTOCODER used on IBM 1401 computers and PLAN used on ICL 1900 series of computers. Third-generation languages are procedure oriented and machine independent, ie a program written in such a language can be executed on any computer, provided the software is available to translate the program into the machine language of the respective computer.

Using *high-level languages* the programmer concentrates on the problem to be solved rather than on the design features of the machine to be used. He describes the procedure for solving the problem in terms of the natural steps used by humans. Each step in a high-level language may be translated into a series of steps in the machine language before execution by the computer. Examples of high-level languages are BASIC, FORTRAN, COBOL and Pascal. These languages have been standardised by the International Standards Organization and are widely used.

Third-generation languages still needed vast numbers of lines of codes for typical commercial programs. They were designed for data processing professionals rather than for end-users. It was very time-consuming to test the accuracy of programs (to 'debug' them). Modification of complex programs was very difficult and huge investments went into maintaining them. A variety of software development tools have now been developed to solve this problem. These tools range from simple query languages to application program generators (APLs). Such tools are referred to as fourth-generation languages or 4GLs. In addition to employing sequential statements, as do third-generation languages, they employ a diversity of other mechanisms such as form filling, screen painting, questionnaire with menus, commands, etc. The focus in these languages is to specify what is required to be done rather than how it is to be achieved. If the problem can be specified in computable specifications, machine code can be

generated from it mechanically. A large number of such tools are currently available; for example, ORACLE, PROGRESS, SEACHANGE, etc.

EVOLUTION OF OPERATING SYSTEMS

Early computers could handle only one program at a time. The operator had to perform a number of tasks manually between every two jobs. The job program and input data would be loaded on input devices, the storage areas in the processor would be cleared of any data remaining from the previous job, appropriate switches would be set, and the job would run alone in the processor until it was completed. After completion, the job program, input data, and output results would be unloaded by the operator and the entire ritual would begin again for the next job. Because the computer sat idle while the operator loaded and unloaded jobs, a great deal of processing time was lost. The situation was further degraded when a fault occurred during the running of a program. The operator had to collect diagnostic information in an attempt to pinpoint the fault. This was very time consuming.

The operating system (OS) software was designed to perform routine housekeeping operations and to manage the computer resources for their optimal utilisation. The early mainframe operating systems reduced the idle time by allowing jobs to be stacked up in a waiting line. When one job was finished, system control would branch back to OS software which would automatically perform the housekeeping duties needed to load and run the next job. This automatic job-to-job transition is still one of the major functions performed by a modern mainframe OS.

The operating system has also made it possible to process more than one program concurrently on the same computer. This feature is called 'multiprogramming'. The input and output operations in any computer are generally many times slower than the processing speed in the central processor. Thus when the computer is busy executing only one program, the processor has to wait for long intervals when the data is being read into the memory or the results are being transferred to an output device. This considerably reduces the processor utilisation. The operat-

ing system software can switch control between programs so that each program is processed for short periods in an appropriate sequence. The program that has to wait for an input/output operation relinquishes control in favour of other programs until it has completed data transfer.

APPLICATION PACKAGES AND DEVELOPMENT AIDS

While system programs such as compilers (language translators) and operating systems are designed to operate, control and extend the processing capabilities of the computer itself, an application program is needed to solve a specific data processing task for a user. Conventionally, application programs are custom-made according to the specific processing requirements of the user. This requirement has often involved heavy investment and long development periods. Internal expertise was required in the organisation to develop such programs or at least to maintain them. With the proliferation of computers, multiple users trying to solve similar problems on computers became common. To meet this situation, independent software suppliers started developing application packages to supply them as prewritten programs to prospective users.

Application packages are designed for use in more than one environment or organisation to perform specific functions which are common to these organisations and are performed similarly, eg accounting, payroll, sales order processing, etc. The functions performed by the package may be generalised to meet the requirements of a large number of organisations but parameterised to cater for the individual needs of each organisation. The cost of developing the package gets distributed over the user organisations. The application packages involve low capital cost, are available for immediate implementation, and eliminate the need for internal expertise of developing and maintaining software.

A large variety of application packages are currently marketed. These include: basic accounting functions (general ledger, accounts receivable, accounts payable, payroll, fixed assets, etc); manufacturing operations (bill of materials, inventory control, materials requirement planning, production scheduling, cost

accounting, etc); distribution operations (order entry, sales analysis, route scheduling, etc); finance functions (budgeting, projections, economic analysis, etc); and a host of other functions. In addition, general-purpose application packages like spreadsheet, word processing, data management, office systems. etc can be used for a wide range of tasks.

A number of software tools have been developed to support and/or automate the process of software development. Significant among these are the 'code generators' and 'application generators'. Code generators produce applications in the form of a high-level language program (or possibly a low-level language program), usually for subsequent translation and/or execution independent of the parent tool. Therefore the application is a separate piece of stand-alone software which could have been produced by hand-coding techniques. Application generators, on the other hand, incorporate a translator and also control execution of the generated application. Hence the generator and the application are both required to be present at run time. The code generator produces commonly used high-level languages such as COBOL or BASIC. The application generator either has an internal translation facility which generates the machine code or, more usually, interprets parameter tables at run time. The user feeds the requirements through an interactive terminal under the control of the tool, by the methods of form filling, screen painting, questionnaire or a set of commands.

Screen painting uses a video display screen to simulate a screen or a page which is to be developed for an application. The generator allows the developer to configure the layout of the applications display or page (for a report) by presenting a blank screen initially on which the developer 'paints' or places, using cursor commands or keys, the required information in the appropriate positions.

Form-filling (or fill-in-the-blanks) is another widely used technique for accomplishing a software component definition. This method also assumes that the generator utilises a visual display unit, and in this case the screen presented to the developer is not blank, as for screen painting, but is pre-formatted with available options.

The *questionnaire* approach again presumes that the developer is working at a visual display terminal. The particular aspects of the application under development are presented in the form of a series of questions requiring usually an affirmative or a negative response from the developer. Otherwise, it may present a set of options in the form of a 'menu' to the developer who may select the option(s) from the menu by depressing selected keys.

The *command language* approach is the same as that used in conventional high-level language programming except that the commands used are likely to be less procedural. This method can be used for any of the components of an application software package and can also be used off-line (non-interactively) as well as on-line (interactively); so in that sense it is a very flexible method.

DEVELOPMENTS IN TELECOMMUNICATIONS

Telecommunication systems such as telephone, telegraph or radio transmission systems have existed for many decades. The early communication systems used *analogue* signals to carry information. An analogue signal is continuously variable: the waveform is analogous to the information being transmitted. For example, the loudness and the pitch of the speech are reflected in the variation of the amplitude and frequency of the corresponding speech signal. An analogue signal is generated by an analogue device such as the microphone in a telephone handset.

Analogue transmission is not very suitable for transmitting data over long distances. Firstly, the transmission environment is sensitive to outside interference which affects the signal waveform and distorts the message being transmitted. Secondly, analogue signals need to undergo staged amplification to restore signal strength when transmitted over long distances and this creates a cumulative build-up of errors due to noise interference.

The need for utmost accuracy in data communication led to the concept of digital transmission. Digital transmission is far more tolerant of the effects of outside interference because the receiver simply needs to distinguish between discrete signal levels. Digital transmission also makes more efficient use of the transmission medium so that many times more information can be sent on the