OCCASION

This publication has been made available to the public on the occasion of the 50th anniversary of the United Nations Industrial Development Organisation.

DISCLAIMER

FAIR USE POLICY

CONTACT

Please contact publications@unido.org for further information concerning UNIDO publications.

For more information about UNIDO, please visit us at www.unido.org

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
Vienna International Centre, P.O. Box 300, 1400 Vienna, Austria
Tel: (+43-1) 26026-0 · www.unido.org · unido@unido.org

18175

Distr.
LIMITED

PPD.99
19 December 1988

UNITED NATIONS
INDUSTRIAL DEVELOPMENT ORGANIZATION

ORIGINAL: ENGLISH

THE SOFTWARE INDUSTRY: DEVELOPING COUNTRIES AND THE WORLD MARKET*

Prepared by the

Regional and Country Studies Branch

Industrial Policy and Perspectives Division

*The designation employed and the presentation of the material in this document do not imply the expression of any opinion whatsoever on the part of the Secretariat of the United Nations Industrial Development Organization (UNIDO) concerning the legal status of any country, territory, city or area or of its authorities, or concerning the delimitation of its frontiers or boundaries. Mention of company names and commercial products does not imply the endorsement of UNIDO. This document has not been edited.

V.88-31689

Contents

INTRODUCTION

This study analyzes the software industry from the point of view of a new entrant to what is now a complex and world-scale market. A new software producer faces new opportunities and difficulties, and to be successful requires a wide variety of skills.

Two main trends dominate the industry. The first is the growth of software as a traded product, rather than an individual service. Software is designed, produced, packaged and marketed on a large scale and in a sophisticated way.

The second trend is the influence on software of technological change in hardware. In general, hardware developments have moved ahead of software's ability to exploit them fully. The variety and pace of hardware change means that many opportunities exist for new software products.

UNIDO has for some time been emphasizing the importance of the software industry for developing countries, and the need to incorporate an awareness of its significance in national technology policies and programmes. UNIDO's Technology Programme has produced a number of documents dealing with several technical aspects of software and its production, and detailed legal and institutional questions have also been a particular focus.*/

However, the present study focuses on the techno-economic environment in which software production takes place, attempting to analyze some of the forces at work which create both difficulties and opportunities at the level of the individual firm in the industry. The emphasis is on the mix of forces, the interplay between hardware and software trends, both technological and commercial, the role of industrial policy in helping or hindering growth in the industry, and the characteristics of a successful strategy at the level of the individual firm.

The study begins with a review of concepts and definitions. Section 2 then analyzes the basic actors, the groups at work in the industry, and the roles they play. Section 3 looks at recent trends in hardware, concentrating on the relationship with software trends, the more significant of which are examined further in section 4.

Section 5 concludes the study with a summary of findings in the form of strategic considerations for software producers.

---

*/ See, for instance, C. Correa, "Trends in commercialization of software in developing countries" UNIDO/IS.574, H. Kopetz, "Guidelines for software production in developing countries", UNIDO/IS.440, and, most recently, H.-J. Schneider, "Software Production: Organization and Modalities", UNIDO/IPCT.63. A regular coverage of software is also found in the UNIDO Microelectronics Monitor.

1. SOFTWARE: ITS NATURE AND SIGNIFICANCE

1.1 Definitions of software

Software is a set of instructions to a computer. It is thus distinguished from for instance hardware, which is any component of the computer itself or data which is what the computer uses or generates to make output, which itself is the reason why the software is supplied to the computer.

This definition, though of course it is not a rigorous one, has the major advantage that it focuses on the use of computers as a process, and makes a distinction between the ultimate product and the means of transformation. Many alternative definitions exist such as that "hardware is what you can touch, and software is what you can't touch". This however does not make clear the distinction between software and data.[1]

The first OECD study on software adopted a definition which also follows this approach, referring to "... a combination of data and instructions, many being algorithms ...".[2] In the present study, however, we prefer to adhere to a narrower definition both because it delimits to some extent what is still a very large topic and also because of the quite different market forces at work.

The same OECD study also provides definitions from the World Intellectual Property Organization (WIPO) and from the International Standards Organization (ISO). The WIPO definition appears to be rather broad, since it includes not only computer programme but also descriptions of them and instructions for their use. The ISO definition also includes documentation, and contains the important additional statement "Software is independent of its carrier media."

At first sight the last statement appears unexceptionable, and seems to be in conformity with the "intangibility" quality mentioned above. But it raises some difficulties in the detail consideration of software and especially its commercial aspects. In practice, software is not independent of its "carrier media", and the disembodied view of software is inadequate for an understanding of the ways in which developments in software are related to those in hardware, and of the ways in which software is marketed.

1.2 The significance of software

Whatever precise definition is used, the idea of a set of instructions is almost always understood as the underlying nature of software. Instructions are commands to the hardware, and at a fundamental level of view it can be seen that the hardware determines what software is written. This is because the set of all possible instructions is determined by the hardware: there is no point in supplying an instruction that cannot be acted upon by the machine which receives it.

The basic instructions which the machine understands (and is built to understand) are called the "machine language". They can usually be represented by binary digits, i.e. a string of 1s and 0s; perhaps by hexadecimal codes (numbers to the base 16). In any case, manipulation of these codes is usally simplified by a piece of software which allows the programmes to use either mnemonics such as ADD or STORE rather than a string of difficult-to-recognize digits. Such a programme, usually called an assembler would in turn be used to write another programme which would allow more powerful commands, and so permit the instructions for mathematical or

other manipulations of data to be expressed in a form nearer to the formal way in which such manipulations would be described in speech or in writing.

The set of such formal instructions is usually called a high-level language. Examples include COBOL, FORTRAN, C, Pascal, etc., and they are discussed in more detail in Chapter 4 of this study. They are mentioned here only to show the way in which software in one sense is indeed independent of the hardware. A definition of C, for instance, usually says nothing about the type of computer on which it can be used. In general, C as a concept is hardware independent, just as most other high-level languages are also.

However, for a computer programmer actually to use C on a computer, in the sense of running a programme written in C and obtaining results from the data being analyzed, there has to be a means of converting C instructions (usually called statements) into the machine language of the computer concerned. As noted above, this means of conversion is in fact another piece of software, and it is usually called a compiler, since it takes all the C statements and converts them to a set of machine commands. (Another sort of translator is called an interpreter: it converts line by line). Unless a compiler (or interpreter) exists, C cannot be used on the computer in question. Thus although in theory a piece of software may be machine independent, it is not really so in practice.

There is a further problem which occurs not only with software but with data also. This relates to the reading by the computer of software and data, usually in the form of magnetic media such as diskettes and tapes. These themselves are of course hardware, but the way in which the software and data are stored on them varies much more than the hardware does. Thus a piece of flexible plastic in the form of a disk, wrapped in a flexible cover, may be identical for many different micro-computers, all of which are capable of understanding the software written on it. And yet each of them may use a different code to represent the characters and, more probably, each will also use a different way of laying out the data and software on the disk. Some might use one number of divisions, some another. Some might use concentric ring patterns and some a spiral pattern.[1/]

The role of software is not only, of course, in computers per se, even though these are being used as examples in the present discussion. Software is the set of instructions stored in anything that is programmable, and this includes many different kinds of machinery and equipment, from machine tools to telephone and communications equipment and dispensing and vending machines. Increasingly such machines embody software to control their operations, not only because it is much easier and cheaper to solve many problems in terms of programmed microelectronic circuitry but also because it makes it much easier to make changes in the way the machine operates and thus to adapt it to different uses.

The case of numerically-controlled machine tools is a particularly important one: it is the spread of microelectronics which has allowed these to become even more flexible and powerful, even though the basic ideas go back to the punch-cards used to change control of weaving equipment in the textile mills of the last century. The programmable character of modern machinery means that it can more easily be adapted to changes in the design or the content of the product being produced, thus allowing for very rapid re-tooling in the face of changes in demand patterns, competitive pressures or changes in primary input costs. Software thus is a usually hidden but essential element in many aspects of manufacturing. The spread of robotisation, factory automation, computer-aided design and manufacturing (CAD/CAM), and computer

integrated manufacturing (CIM) means that software will have an even more important role in the future, especially in the linking of the software embodied in individual machines into a fully communicating and controlled system of production.

It is, however, software for computers which is most easily discussed, especially because here is most easily explained the concept of software as a traded commodity.

1.3 Software as a commodity

The International Standard Industrial Classification (ISIC) of the United Nations is intended to cover all economic activities, and being based on a structure first conceived in 1948, does not cover the production of software anywhere. Equally, the United Nations Standard Industrial Trade Classification (SITC), Revision 2, does not deal with software at all. This is more surprising because it dates from 1975.[4] The exclusion of software from classification systems may be partly because of its (relative) newness, but a major reason must also be its intangibility as referred to earlier.[5] The easy option may have been taken of treating the software industry as a service with no permanent tradeable outputs. This approach may explain its exclusion from SITC, and its brief history may be the reason why it is not found in ISIC.

However, the fact is that software is now an industry of global reach, whose products are conceived, designed and marketed in a typical manufacturing-type operation. The writing of software to order still continues extensively and will continue for a long time to come.[6] But increasingly it is the mass production of software as a tradeable good which represents the visible shape of the industry.

Complex issues arise, not just at the level of statistical classification, but at the very practical levels of customs tariffs, cross-border data flows, patents and copyright law. More fundamentally, they arise in the questions of the nature and future shape of the industry, the evolution of the manufacturing process, and the internationalization of production. So rapidly is the role and nature of software developing that a fully comprehensive view of its scope is not easy. However, certain characteristics are visible which suggest that it can usefully be regarded as an industry producing merchandise which is traded. This aspect of software is growing rapidly. The industry is labour-intensive, but also skill-intensive. In addition, technological change affects this industry also: there is a tendency towards increased capital inputs, although clearly the overwhelming character of the industry is that of a (skilled) labour-intensive one. Technological change has also affected product life cycles, which are continuing to shorten, but perhaps the biggest influence on this tendency has been the increasingly fierce competition among producers. A search for new product ideas is a continuing necessity for survival.

With respect to technological change, the time needed for most innovations is substantial. This is because of the labour- and skill-intensity of software development. Thus, between the conceptual definitional stage of the new software and its fault-free, saleable form, a considerable time, sometimes years, may be needed. New tools may allow for a shortening of development time, both by increasing programmer productivity and by simplifying the complex managerial tasks associated with large software projects. However, one interesting effect of the long product development times for new technological levels of software is that the future shape of

software can be predicted with more confidence than for other products of industry. Thus, areas of search for new opportunities can be more easily identified. In general, the software industry, although many large firms are found, remains open to newcomers. It offers potentially large rewards for innovation. Because of constant change in technology, it provides many niches for which distinctive products can be developed.

1.4 Software and developing countries

For developing countries such considerations are particularly important. Software skills are in any case necessary to maintain or improve the competitiveness and efficiency of all sectors of their economies, not just manufacturing. However, only mastery of such skills can allow the developing country to control the direction of its informatics development as a whole, since reliance on imported software will direct its production structures into ways of doing things which are determined elsewhere. This can be a difficulty not only from a cultural point of view. It can raise problems also in very practical ways. Imported software may require data in a form in which it is not usually kept in the country, or may assume a particular system of accounting or taxation which is usual in developed countries.[7] To use them may mean a complete alteration of patterns of work and ways of thinking. Often the resulting confusion could wipe out any gains in efficiency made by computerization. Even if this is not the case there is still the problem of interfacing with other, manual systems which use the accepted national approach to the problem in question.

The mastery of software skills becomes especially critical if the informatics field is recognized as a main means by which developing countries can increase their international competiveness. The developing countries' cost advantages of low wages can be and in fact are being quickly eroded by heavy investment in automation and CIM in developed countries. Developing countries in any case face enormous problems of co-ordination in enabling their economic and social systems to improve the living standards of their people, and informatics technologies can contribute significantly to their solution.

These arguments for informatics development in developing countries are well known. They are restated here because an additional point is to be made, which is that, in evaluating the scope of software development in a developing country, the external market has to be considered.

The export of software (which can be to another developing region or the world market as a whole) has distinct advantages for developing countries which require that it be seriously considered as a policy option. Firstly, it should be recognized that software is an industry like any other with advantages (and of course disadvantages) as a component of an export strategy. But secondly, there is the vital point that a concentration on exporting software will promote quality and technological progress in the domestic industry in general, in a way that concentration on, for instance, clothing manufacture or food processing will not. This is because competition in the latter field is based in so many cases on price alone, and in software exports this can never be the case.

The improved quality of a software industry oriented to the external market means in turn that the rest of the developing economy concerned then has direct access to strategically important components for improving organizational efficiency, increased productivity, enhanced competitiveness, and the flexibility needed to respond to rapid changes in economic conditions

The increased application of microelectrcnics, computers, telecommunications, and automation all require a mastery of software skills. A competitive software industry in a developing country, which succeeds in selling software products in a very challenging and rapidly shifting world market, is a national asset which can contribute to most other economic activities within the country.

2. OVERVIEW OF THE INDUSTRY

2.1 Types of software

In this section, we attempt a statement of the kinds and classes of software that are marketed, not in attempt at a definitive taxonomy, but in order to provide the working language needed for a discussion of the actors in the software industry, the roles they play and the strategies they follow.

As noted above, we are restricting our working definition of "software" to include only instructions to the computer or other programmable device. This, in fact, could make our definition of software the same as that of "programmes", but it is probably better to regard software as a generic term, covering all traded software. The units of software may be a programme or a set of programmes (sometimes called a "suite") if they are related to one another.

Software can be written by the user ("developed in-house") or it can be acquired from an external supplier. Sometimes this doesn't cost anything. This can, of course, be the case if the software is illegally copied. But it can also be the case if the software comes from a university, for instance, or a voluntary group of users of a particular type of computer, or a professional society. This is called public domain software and is particularly important for microcomputer software (such software can often be worth investigating, before embarking on a major software development project).

In turn, the software acquired externally can either be written to order to meet an individual requirement ("custom software") or it can be already written and on general offer. This is called packaged software.

The main types of software are systems software and applications software. Systems software is the software which helps the computer to work, and applications software is the software which does the work which makes the computer useful. Thus, the payroll programme, which calculates the monthly salaries to be paid, is an applications programme, while a disk utility programme, which organizes the various files of information in some way on a magnetic storage device, is an example of systems software.

Systems software includes, most importantly, the operating system of the computer. This is the software which governs the interaction of all the components, interrogating the operator's keyboard and writing to the screen scheduling tasks, deciding on priorities, organizing storage, etc. Operating systems can be very rudimentary or extremely sophisticated. They are crucial in that they usually determine the environment for any other software to be used on the computer. An operating system is almost always present, and any other software to be used on the computer has to be compatible with it. Thus, to take an example, MS-DOS is an operating system for computers of the IBM Personal Computer type, and the software to be used with it is said to "run under MS-DOS". If an operating system is sufficiently sophisticated, there may be no other systems software available or needed for the computer, but more usually there will be so-called utilities to improve upon the tasks carried out by the operating system or to supplement them. Another example of systems software is a special programme to handle communication with computer terminals or with other computers or devices.

Application programmes include packages for automating the calculations involved in almost every field of human activity. A package is a term applied to a more or less stand-alone computer programme which offers full control of

input and output and of storage within the programme. It will typically be able to carry out a wide variety of different tasks within a subject area. No hard and fast rules separating programmes from packages have been drawn up, but the ability to issue alternative commands to a programme may be the key characteristic that determines whether it is a package or not. Thus, a spelling checker is probably a programme: it takes a file of text and compares each word against a dictionary to find errors. But word-processing software is a package: there is usually a sufficient number of choices of things to do, e.g. to edit, merge, cut, paste, search, word count, etc. (There will probably be a spelling checker incorporated, also).

As noted above, "packaged software" is something rather different. Here the word package is used to emphasize that what is being sold is being treated as a consumer good, a product which could be found on shelves in shops. The software (usually on diskette, with a user's manual, wrapped in a box, is marketed often with the same skills as are used to market any other fast-moving consumer good).

The distinction between systems software and applications software is not always absolute. A database package, for instance, is systems software in that it is concerned with the organization of data and its storage, and it may arrange all the data on the disks of the computer, superseding the existing arrangement followed by the operating system. Similarly it may carry out communications and other functions normally carried out by the operating system. But a database programme, from another point of view, is applications software, because it usually provides tools for sorting, selecting, totalling, and, in general, what is called report generation, i.e. the carrying out, as required, specific analyzes of the data in the database and the presentation of the results in an acceptable form. Computer language software is also difficult to classify: arguments for both points of view could be found.

But the distinction between systems and applications is still a useful one, and will be followed in this paper. There is plenty of software which is unambiguously of one kind or the other, and the classification helps to analyze the industry in more detail, and its relation to the hardware question. Secondly, the two types of software are increasingly being sold to distinct markets. Systems software is largely bought by computer specialists and professionals, and applications software more by non-specialists who are interested in the computer only as a tool to assist in accounts, engineering, medicine, law, customs clearance or some other activity. It has been increasingly noted that even though the actual purchases of applications software may be made, in a company, by the data processing department (i.e. the traditional computer users) the choice may be as a result of pressure from the non-computer staff who have identified the piece of software as useful for their own work. The so-called end-users thus exercise increasing influence, and marketing efforts are increasingly directed towards them.

2.2 Software producers

The main types of producers are as follows:

- Computer manufacturers
- Software companies
- Original equipment manufacturers (OEMSs)
- Value added resellers (VARs)
- System houses
- Computer users

The computer manufacturers are major producers of software. In the past, computer manufacturers used to provide almost all the software that was available for the computer in question, apart from that written by the purchaser. There was thus no so-called "third-party" software available. Manufacturers have to ensure that there is enough software available to encourage purchase of the machine, and this they do either by following an existing hardware design, or by providing a version of a standard operating system, for both of which sufficient software is considered to be available.

Software companies are difficult to classify. They range from very small enterprises, with perhaps just the proprietor writing and selling the software, to large multinationals with complete international distribution systems and full service and support networks. The common characteristic of software companies is the high ratio of skilled employment to fixed assets. Certainly the company will have computers on which to develop the software, but the main assets are its people. The R&D phase is the crucial and the expensive one: production costs have been low up to now, involving merely the duplication of a tape or diskette. However, increased competition has led to a more demanding market for documentation. The companies have to provide detailed manuals and usually spend a good deal in making them attractive. The need to prepare different language versions is a further cost of production. In addition, sales and marketing expenditures in a highly competitive field have become increasingly significant. Also, apart from documentation per se, the purchasers of the software will expect, depending on its price, a range of services to be provided by the software producer. This includes telephone support (such as a "toll-free hot line") which provides the user with help and advice. Other services provided can include newsletters, user groups, etc. While the user may be charged a fee for some of these, there is usually a subsidy from the software producer also.

The name OEM is misleading, since what OEMs do is to put together a package of hardware and software, put their own name on it and sell it. Thus, an OEM might buy a basic computer from one manufacturer, memory and peripherals from another, and software from a third. The OEM may, however, also produce or commission extra software. The whole package is then marketed as if it were the OEM's own product.

A VAR (Value added reseller) is rather different, being typically a dealer who buys in all the hardware often by arrangement or exclusive agreement with a single manufacturer, and then sells it on, with perhaps proprietary software and training included. The VAR will specialize in a particular application area, such as accounting, architecture, etc.

A system house is a more autonomous body because it does not carry out purchase and resale activities. It is exlusively a service sector body. It can advise on purchase, design systems, and carry out the necessary programming. This software production is thus typically done to order, meet the needs of a particular client. It may, if successful, also lead to the production of a package sold again to other customers. The transition from system houses to software companies is thus by no means unusual.

The computer user is the remaining category of software producer, which is by far the largest. Computer users, especially of mainframe computers, often develop large quantities of software for use within the organization. In fact, this is the overwhelming majority of all software ever produced. Little of it, however, ever moves outside the organization in which it is developed. There are exceptions, however, and a company, for instance, in a quite different field may develop software that is marketable. In so far as

its release does not reduce the competitive advantage of the firm it can be a profitable subsidiary activity. For instance, the McDonnell Douglas Company, which makes aeroplanes, has developed a number of software products which were originally for its own use in engineering design. It now sells these and has moved into many other computer fields also, being a system house as well as software producer. Many other examples can be found of the transition of the computer department of a company becoming an autonomuous system house and software producer. The phenomenon is more common in the services sector, however, with banks, insurance companies, and accounting firms being typical breeding grounds for such activity.

2.3 The size and structure of the software industry

Assessment of the size of the software market is very difficult, as is the estimation of the value of software production. The latter question is particularly complicated due to the fact that, as noted above, most software is not traded, but developed within companies or institutions for their own use.

Even considering only commercially handled software, however, the statistical classifications used in industrial statistics are usually inadequate. A particularly intractable problem is caused by the fact that much traded software is produced to order by system houses who will charge the customer not only for the software itself but for a range of consultancy services associated with it, including system analysis and general business services. If the system house is a computer bureau also, it might provide, as part of a package, the actual data preparation and processing also. To separate the specific software costs is not easy. A similar problem exists with what is called "bundled" software, which is software sold with a computer as a package deal, or provided free to purchasers of a particular computer. International trade in software is equally difficult to measure because of the similar lack of adequate classifications as well as the different treatments of software for custom valuation purposes. The ease with which software can be transmitted by telephone lines, for instance, further complicates the measurement of its international trade. Finally, the problem of software piracy makes it difficult to envisage any fully statistically consistent pictures of production and trade in software. The spread of pirated software is due principally to the increased demand for packaged software for the millions of personal computer users.

In spite of these caveats, however, estimates of the value of software can certainly be found. The OECD has been particularly active in this field, and published a first survey in 1985. It drew on a wide variety of data sources but was unable adequately to separate out specific software activity from the services provided by system houses. A new study soon to appear gives some alternative estimates.[8/] Broadly speaking, the more country coverage is increased, the less precisely is it possible to confine the figures to software alone. Thus, for a total of 32 countries, the software market in 1984 was $US 26.6 billion, and in 1987 estimated for 29 of these countries to total $US 48.8 billion. The country coverage could be wider, but then it would not be possible to separate out all software from services or hardware sales.

United States estimates give the world software market as having a value of $US 30 billion, of which United States suppliers have an approximately 70 per cent share. Packaged software revenues amounted to 63 per cent of the United States total revenues in 1985, and packaged software for personal computers was the fastest growing segment. Other estimates include those of United States revenue from overseas sales of software being over 20 per cent of the total revenues, thus amounting to about $US 4 billion. Packaged software is 30 per cent of the United States total exports. The largest software markets for United States exports are given as Canada, Western Europe, and Australia.[9/]

Another world estimate comes from an authoritative private source, and gives "software costs" as "$US 140 billion worldwide" in 1985. The figure is given in the context of a discussion of software productivity, and clearly includes in-house development of software. The writer refers to a present growth rate of 12 per cent per annum, and projects the world total to be $US 450 billion in 1995, with the United States share remaining at 50 per cent.[10/]

Perhaps because of the statistical difficulties in coverage of the total software market, it is easier to obtain estimates of packaged software alone. Thus, one estimate gives a world market projection os $US 22.3 billion in 1989, having been $US 5.8 in 1984. Of this packaged software, the share of applications software is 72 per cent in both years.[11/]

The structure of the industry is a complex one. Certainly there are some large firms, and the tendency in recent years, especially in the United States, has been for them to expand further by acquisition. In the first six months of 1987, there were 137 acquisitions or mergers in the computer services field (here including companies in the software business). The associated value of these agreements amounted to $US 2.1 billion. In the previous full year, by contrast, there had been only 130 such arrangements, with an aggregate value of $US 1.9 billion.[12/]

There are reasons for this tendency: the principal ones being the human capital assets of software companies. A team of unique talents will be attractive for a takeover because they will bring benefits to the acquiring company in a way in which a simple increase in recruitment would not. In the specific case of packaged software, a takeover will allow the acquisition of product for which the development of something comparable would take years and for which success might be doubtful. But the single factor most influencing the tendency towards mergers in the software business is that those within it are best placed to identify opportunities. A software company is more attractive to another one than to a financial conglomerate.

2.4 The role of standards

Two types of standards are found in the software industry. The first is the formal standard established by a national or international body. The second is the *de facto* standard, which is a result of a particular piece of hardware or software becoming so widely used that it is recognized as having created a market. Examples of formal standards include those adopted by such bodies as the American National Standards Institute (ANSI) and those of the International Standards Organization (ISO). In many cases, and particularly with ANSI standards, a national standard goes on to become internationally accepted, with perhaps a version of it subsequently being adopted formally by ISO.[13/] Examples of *de facto* standards include MS-DOS and the related PC-DOS, both developed by Microsoft Corporation for the IBM PC and compatibles

These are operating systems, and therefore they effectively determine the environment in which software to be used on these computers has to be written. While being extremely widespread and having sold in the millions, these standards have not been adopted by any body as such. Yet they have influenced the creation of thousands of other software products intended to be sold to those who use this operating system on personal computers.

A software standard is a definition of a concept, to be followed by those who implement it as software. Thus, the standard for a programming language is a full definition of the language, its structure, its grammar and syntax, its statements and commands. A software producer can produce a compiler which exactly reflects the idea of the language, and can then describe it as "a full implementation of the XYZ standard". Another software manufacturer can produce a piece of software written in that language, using only that precise version of the language defined in the standard. In principle, therefore, the customer can buy the language compiler from one company and the application programme from another, if they both follow the same standard. If they do not, however, and therefore, if the customer has no guarantee that they will work together, he may decide to buy neither. Languages and application software are a simple example. In practice, most application software is not distributed with the so-called "source code" because this is often regarded as proprietory information of the vendor, and to allow other to have access to it would allow them to modify it rather than coming back to the original vendor for help, and thus further business. It would also allow other companies to imitate some of the particular tricks and embodied skills within the software. More practical examples will be found in the area of compatibility between application software and operating systems, or between two application programmes. With respect to data compatibility, for instance, the highly successful package LOTUS 1-2-3 stores its data in computer files in a certain format. Another software manufacturer may advertise his product as able to read files in LOTUS format, thus, this product will be immediately attractive to those who already have the LOTUS product. There is a curious secondary effect, in that the very fact of a manufacturer proclaiming a product to be compatible with LOTUS generates a further support for LOTUS as a standard.[14]

In general, the issue of standards is very important for software producers, particularly those beginning in the business. To have a clearly defined standard in the area of application can be very important for the producer, because it provides him with some sense of the market for the product, and it provides him with a relatively stable technical environment in which his software development can take place. He can be reasonably assured that if his product is a failure, it will not be because of its technical unsuitability, but for some other reason. In principle, therefore, standards can be just as useful to the software producer as they are to the software consumer. However, it is a fact that the setting of formal standards is a complicated process of national and international negotiation, involving many scientific committees at different levels. Typically, the major producers are represented on at least some of the committees involved. They are in a position to influence to some extent the adoption of the standard in such a way that it includes areas in which they have particular competitive advantage or particular skills. In any case, by being involved in the decision-making process of the setting of the standard, they have access to information as to the form the standard is likely to take. This gives them a competitive advantage which is not enjoyed by the small and new producer of software. From this point of view, therefore, software standards are a disadvantage as far as the new producer is concerned and have severe draw-backs for him or her. However, it is clear that the benefits of standards outweigh the disadvantages. The question must therefore rather be how best to mitigate the

disadvantages from which the small producer suffers. As has been seen, the setting of de facto standards is something with which the small producer is not involved at all and his exploitation of the market opportunities they create is even more difficult than with formal standards, since detailed information on the product which is creating the standard may be very difficult to come by.

3. THE EFFECTS OF HARDWARE TRENDS

3.1 Processors and memories

The processor is that part of a computer which carries out the programme instructions which constitute software. Thus it will typically carry out arithmetic tasks such as addition and multiplication, together with other actions such as comparison, branching and looping. These latter and similar tasks are particularly important because they allow the machine the flexibility in hardware terms that is consonant with the generality of software.

The kinds of tasks that the processor can do vary widely in number. Two seemingly contradictory trends exist in this field. One is to make more and more sophisticated processors capable of a large number of tasks. This is supposed to make software development easier, since the machine language programmer then has access to a number of powerful tools. The other trend is to reduce the number of instructions understood by the processor. Broadly speaking, the fewer the number of instructions intelligible to the processor, the faster that processor can work, or perhaps more correctly, the faster that software written for that processor can work. Such processors with reduced instruction sets are called RISC processors.[15/]

The microprocessor is a processor on a single chip. The design trends noted above are intended to contribute to the speed of computing. Another way to do this is to speed up the processing itself, and a variety of technological approaches are used for this. Increased integration of itself reduces the delay in instructions being acted upon. The selection of new materials, such as gallium arsenide can also increase the speed, but this means new approaches to manufacturing, and gallium arsenide technology, because of its cost, has up to now been mainly confined to military applications. Increasing the amount of information that the processor can handle at any one time, its basic working unit, is another way of increasing the processing speed. Thus, the earliest microprocessors were 4-bit (such as the Intel 4004), and these have been succeeded by 8-bit (such as the Zilog Z80), 16-bit (such as the Motorola 68000) and 32-bit (such as the Intel 80386).[16/] Thus, in terms of internal architecture, microprocessors have reached the same level as minicomputers, for which a 32-bit processor is usual. Another aspect of this is that it makes it easier and thus quicker to handle large amounts of computer memory.

Not only has the facility with which large amounts of memory can be handled increased, but the speed of the memory (i.e. the speed with which information can be stored or retrieved) has also grown. Allied to this has been a fall in the unit price of main memory, a fall which has been steady for many years. A temporary shortage in recent months has caused a departure from this trend, but the underlying tendency continues to be downward, an inescapable consequence of technology development and the search for a differentiated product, with both these taking place in a fiercely competitive environment.

The consequences for the software market have been very striking. In fact, software can be said to have been struggling to keep up with what has been happening in the hardware field. The capabilities of new computer systems, given the processor and memory trends described, are of a different order to most of the software available for them. This is partly because of the general need to have access to the hardware in order to develop software that fully exploits it: this software development can take several years.

However, a trend in software is towards more "user-friendliness". This means, in general, that computers become easier to use. More of the user's mistakes are corrected, more "help messages" are given, there is a tendency for the user to be presented with a range of options rather than be compelled to remember a series of cryptic commands. Growing computing power makes it possible to bring all these ideas to reality. A package has to be elegant, slick and well-planned: the "surface" of the software has to be smooth. No matter how good the central idea of the package is, it must still be well-presented and easy to use if it is to have a chance of commercial success.

The availability of faster processors and more memory means that the software in turn must take full account of it and, therefore, as well as "user-friendliness", a second consequence of hardware trends is for software to have more functions or features. The integrated package is one which combines what was previously regarded as separate tasks for software. An integrated package will allow for word-processing, graphic display, spreadsheets, database, etc., all of which used to be separate packages. Even where they remain separate, the competitive package will have extra features: a word-processor will have facilities such as a spelling checker, outlining, etc., for which earlier separate programmes would have to have been bought. The reason for the growth in integration or what is called the "increased functionality of software" is the availability of space and speed in the computer which uses it.

Recently a class of materials has been discovered which exhibit the property of superconductivity, that is, of offering little or no resistance to the movement of electrons at temperatures well above absolute zero.[17] This will have important effects on the future microelectronic components, and it will mean a greatly increased speed of operation for processors and memories, as well as bringing great changes to telecommunications and most other informatics fields.

3.2 Architecture

We have spoken as though there were a single processor in a computer, and that is the traditional picture. The so-called "von Neumann architecture" is the internal design pattern roughly followed since the inception of modern computers, where a single processor performs all the calculations and makes all the decisions, one step at a time. In fact, however, this traditional view has been gradually modified. Mainframes and minicomputers commonly have more than one processor, and even microcomputers although they have one microprocessor will often have other processors to control screen graphics or floating point calculations. However, it remains true that these processors do not have equal status, and that one processor provides the main control.

Recently, hardware development has concentrated on the linking together of several processors, perhaps very many to work in parallel. There are conceptual and practical problems, but many of these can in principle be solved by software. New languages and operating systems can allow for a new kind of computer programming which takes advantage of the parallel processing opportunities the new hardware configuration provides.[18]

A further development is in so-called neural computing, which is an attempt to follow what is believed to be the way the brain itself works. Rather than the binary logic at the heart of present day computers, the target is the multitude of interconnexions such as are found in the brain, and the switching processes based upon the attainment of a sufficient number of control signals at each step. The reason for the interest is ultimately the

hope of attaining in hardware terms some of the useful characteristics of human thought processes. While hardware development has still a long way to go, some of the features can be explored in software terms.[19] In particular ideas of multiple association, such as the human brain is so readily capable of, have already influenced database design, as can be seen in the development of hypertext systems.

Hypertext systems are a means by which information can be stored with arbitrarily complex links between its components. Thus, a paragraph of text, for instance, instead of being stored under one keyword or even ten keywords, could have everyone of its words as a keyword and the relationship between the information contained therein and any other piece of information could be followed there through a chain of links. It is this multiple connectivity of information which is analoguous to the way which the human brain stores information.[20]

3.3 Peripherals

Several striking developments are taking place in the field of peripherals. With respect to mass storage, a notable tendency has been that of the growth of optical storage systems. These are so called from their use of a laser to read and also perhaps to write to a storage medium. The advantages over the conventional magnetic storage systems such as hard disks, floppy disks, tapes, are that they provide a much denser way of storing information, that they are not subject to loss of data from magnetic fields, etc. A single CD-ROM, which is the size of a standard audio compact disk, can hold 600 megabytes of information, roughly the equivalent of 1,500 floppy disks of the standard size. A CD-ROM can be written only in the factory. This makes them best suited for information systems which are relatively permanent in nature, typically including such applications as legal information, statistics, encyclopedias, etc. A more flexible type is the so-called WORM which is an abbreviation for "Write once read many times". This is also an optical storage medium which can be written to once by the computer user. When a WORM disk becomes full, it is simply set aside.[21] The full availability of optical disk storage awaits the arrival of a disk which can be written to and erased by the computer to which it is attached. Such a technology is still being developed, but is expected to be brought to market in the next couple of years. The potential of these storage systems is immense. They offer a means by which enormous amounts of data can be stored and easily accessed by a computer user. This means the disappearance, in years to come, of slow techniques such as storage on magnetic tapes of less frequently used information, a practice which is called archiving. They mean also that it will be perfectly practicable to store in character form every document generated by the business, and thus every piece of information in the business could be searched for and analyzed by computer, inciuding, for instance, all correspondence over a period of as many years as it is desired. The introduction of scanners, which can read the type characters from paper into the computer, means that it will be possible to store all correspondence in this way, eventually including handwritten correspondence.

Such technology will provide enormous opportunities for software developers. The opportunities will lie principally in providing tools for those who use this information, in order to allow them to make sensible use of it. It is one thing to be able to access quickly all of the relevant information, it is another thing to be able to decide what is important or what is not. The user will still have to search through the material available. Hence "intelligent" software which will help the user to make best

use of all the information is a promising field. It represents a qualitative change from the kind of software which manipulates the information, as present day database systems do.

The so-called "user interface" has seen developments in several areas. Particularly important is the growing use of graphics. It can be expected that the screens of computer workstations will become larger, will have higher and higher densities, and will provide more and more information to the user. The growth of such techniques as "windowing" allows for the display of several different information areas on the screen, including information about perhaps different processes under way. The increased hardware speeds mean that the numerical calculations involved in graphical displays can be carried out in more and more detail, and this has led to increased sophistication in CAD/CAM systems, for instance, where diplays which can manipulate three-dimensional images are more and more usual. The software developments associated with this include the growth of what are called object-oriented languages, since the traditional mathematical languages such as FORTRAN often used to produce graphics images, are inadequate for the kinds of sophisticated graphics application now being developed. (A further push towards object-oriented languages comes from the growth in artifical intelligence applications). Thus, hardware developments in display technology have created a significant market for software products, and one that will grow.

Communications between workstations and computers, and between computers themselves is another growth area where hardware developments have allowed for the creation of very elaborate networks. These networks in many cases span countries and continents. Where telecommunications have been liberalized, and this is an increasing trend, the linking of computers has become commonplace. Fibre optics developments mean faster communications of this kind. There are, however, still very significant problems in data communication which the emergence of new standards has not yet overcome. Many different protocols exist, different computer manufacturers will follow different standards, and national telecommunications authorities in many cases restrict the use of standards to those specified by them, or refuse to allow the connection of any equipment to the national network which has not been certified by them. This means that communication software in general has many compatibility problems to overcome. There remains also considerable scope for improved networking standards which would allow for more flexible and faster communication between different systems. An enormous number of niche markets is thus created, since software to overcome particular compatibility problems, or to allow two pieces of equipment to be connected together, may provide a neat solution to what is an otherwise intractable problem or, alternatively, a relatively simple problem which has been ignored by the makers of the equipment. Thus, even the confusion in the computer communications field can bring benefits to the software producer, although it is such a rapidly changing field that these have to be continuously solved in new ways. In the longer term, the trend is towards an ISDN (Integrated Services Digital Network) system, which will be a public network to replace the existing separate telephone and dedicated data networks. This is a tendency which will lead to increased standardization and perhaps fewer opportunities of this kind. However, the process is not likely to be completed in a short time. The spread of the Open System Interconnect (OSI) standard is a further development which will influence communications software significantly in the longer term.[22] Again, the transition periods may offer opportunities for the bridging of gaps.

4. TRENDS IN SOFTWARE

4.1 Overview

This section examines trends in software which are expected to have an impact on world markets in both the short- and the medium-term. Software in general can see a long delay between a theoretical advance and a commercial application. This is for several reasons. Firstly, an advance may simply be of no commercial interest. Secondly, the software advance may be rendered redundant before it can be marketed. For instance, a software advance that speeds up a particular calculation or enables less storage to be used may well be irrelevant in a context of rapidly falling hardware prices. Thirdly, software has to encounter significant resistance in its traditional consumers. Computer staff will have invested several man-years in the construction of a particular application. The new product, even if better and cheaper, may still require conversion of the existing data, and time to be spent in a transition period when both systems have to be kept working, the old and the new. It can appear easier to keep the old system in operation. The situation is worse when the product offers a more dramatic break with the past: it may imply data structures and system procedures which are so different from what has gone before that they intimidate the potential user who has been conditioned by the existing system. The producers therefore cannot move too much ahead of the users.

This is not to diminish the role of innovation in the industry, but it must be realised that bottlenecks occur, and the absorptive capacity of the market can be limited. Good ideas may have to wait their turn, especially if their cost is high either in terms of purchase price or in terms of further investment of own resources by the user.

There is nevertheless scope for innovation in software, especially when it is associated with a hardware innovation. The best example is given by the microcomputer. Its widespread availability generated a number of software innovations, with products being produced which were qualitatively different from mainframe and minicomputer software. Spreadsheets are a product of the microcomputer era, which might better be called the personal computer era. This is not only because of the dominance of the IBM Personal Computer and its clones, but also because the personal character of the computer determined much of the software which was written for it. Spreadsheets are targeted towards the manager, accountant or clerical worker who works with tables of figures. At the simplest level, a spreadsheet will maintain the row and column totals of the table, and can be set to automatically adjusted the totals when any individual figure or group of figures is changed. In practice, spreadsheets have grown more and more sophisticated: the possibility to specify arbitarily complex relationships between table entries and between groups of table, as well as the order and manner in which recalculation can be carried out has meant that spreadsheet commands have evolved into programming languages. Nevertheless, the natural matrix orientation of these spreadsheet languages gives them characteristics unlike any of the widespread high-level computer languages such as Fortran, Cobol or PL/1.

A further genre of personal computer software has been the integrated package, where a number of different functions (spreadsheets, database, graphs, etc.) were combined in one and allowed the user, for instance to move data from the database to the spreadsheet, to make calculations from it and then to draw graphs to display it. Previously these steps would have had to have been carried out through the separate selection loading and running of

these different software packages. The integrated software package allowed the user to computerise all his traditional office activities. A later development was the addition of other facilities, especially word-processing, within the package.

A further feature of microcomputer software has been its "user-friendly" character. This arises from two causes. Firstly, some of the individuals who pioneered the development of microcomputer hardware and software were partly motivated by an individualistic feeling about the role of computing power in general. They did not see it as something centralised and something that should be accessible only to the initiated. A second and now dominating feature is that the commercial packaged software producers realise that they are catering to a market the majority of whose members have either no computer experience at all or else are essentially self-taught, being accustomed only to using other packaged microcomputer software. The consequence is that they must at least maintain if not improve on a tradition of user friendliness. This trend is reinforced by the hardware trends already discussed in Chapter 3, above, where the falling costs of memory and storage devices means that software can be bigger and store more messages to be sent to the user, guiding him or her through the use of the programme or package in question and explaining errors, not just pointing them out when they occur. This trend further exemplifies the need for software developments in some cases to wait on hardware developments, even though the inventive character of user-friendly features may in fact be very low.

It is useful to make a distinction, in examining trends in software, between evolutionary change and new technology software. Evolutionary change in software can be defined as change determined by improvements to previous stages of the software, where the lines of descent are fairly visible. In simple terms, improvements have been made to existing ways of doing things. New technology software on the other hand refers to software developments which are associated with using computers to do things previously done in other ways or not done at all. Here can be included most artificial intelligence (A.I.) applications, such as image recognition, speech processing, computerised translation, expert systems etc. It can be stretched to include concepts of CAD/CAM and Computer Integrated Manufacturing (CIM), which amount to new ways of doing tasks which had not previously been automated. The point to be emphasized is that the first computerisation of accounting, payroll, stockkeeping, etc. took place many years ago. New software developments in such fields as these is evolutionary because it accomodates to some degree at least the systems which are already in existence and allows the exploitation of innovation without the perhaps painful abandonment of existing systems.

The line between evolutionary and other software can be hard to draw in some cases, especially when the software product incorporates features which are partly evolutionary and partly not, such as the use of expert systems in database applications. Why, then, make such a distinction? It is important strategically because different markets are in question and it is important also in investment terms. A new departure in software may, precisely because it starts from scratch, involve considerable inputs to reach production status. Financing the development costs may be painful, since cash flow in the early marketing stages may be slow. The evolutionary product, on the other hand, may represent a repackaging of an existing product or its transfer to another computer system. Software tools such as cross-assemblers may make this process easier. Again the new evolutionary product may be able to take advantage of elements of the software environment in which it applies, and thus make use of improvements carried out by someone else. For instance, the

systems software of a computer often undergoes continuous improvement by the manufacturer. It may contain new and efficient ways of reading, copying, comparing and checking data and displaying it, storing it, and printing it. This sort of software is unglamorous and yet it involves a great deal of detailed work. The astute independent software developer can take advantage of much of this work and thus spare a good deal of the development time by finding out how these parts of the system software work and allowing his own software package to use them. As well as saving time in the development phase, this will very often mean an increase in the speed of operation of the finished programme itself. Again, some parts of the system software may be inefficient and the producer will decide to bypass them, approaching the hardware directly. Such practices, however, have their dangers for the software developer. The reason is that they can tie the product too closely to the hardware of one computer manufacturer, or to one version of the system software, and thus restrict the potential market and increase dependency.

4.2 Operating systems

As briefly mentioned in Chapter 1, the operating system provides the necessary control and communication between the different parts of the computer and its peripherals in such a way that it is accessible to the user. The operating system provides a framework, more precisely a defined software environment, in which applications software can be written. It also carries out a lot of the tasks needed by the application e.g. control of the input and output, and, in multi-user systems, time sharing, and automatic back-up and recovery. This also means that an operating system defines a software market.

The point is worth emphasizing: the same operating system, if it is available on different computers can (in principle) allow an application written for one computer to run on another. Equally the same computer can use more than one operating system, and an application written for one computer will not run on an identical computer if the operating system being used is different. The operating system thus insulates the software developer (to a large extent) from the physical computer. But it makes software development less hardware-dependent only at the cost of making it operating-system dependent. (The role of high-level languages in countering this tendency is examined below).

In general a computer is supplied with operating systerm software included, since a computer without an operating system is of interest to a specialised few. Therefore it has almost always been the case that the computer manufacturer supplies an operating system, usually written to take advantage of whatever hardware features characterise the new computer (but increasingly with an emphasis on distancing the user from hardware considerations per se). At the same time, attempts are made to accommodate earlier versions of the computer and operating system, either by providing that any software written to run under the old system will run under the new one even if it does not use all the new features provided in the latest version "upward compatibility", or else by supplying a set of software to assist in the conversion process, that allows the old software to be automatically changed so that it can function under the new system. This is sometimes called an "upgrade path".

Operating systems are sometimes classified as multi-tasking, multi-processing, multi-user etc. In market terms, the most vital distinction is perhaps between multi-user and single user systems. Broadly speaking, mainframe and minicomputers have multi-user operating systems, either in the

sense that the system can deal with a number of different tasks submitted to it, or in the sense that many users are actually physically linked through terminals to the computer.

Multi-user systems are what are needed in ordinary economic activities. Most of these involve exchange of information between those working in the same organization. If the computer contains the organization's data in a multi-user system then all the staff can in principle have access to the data and change it as necessary. It is this relatively simple idea which lies at the heart of the database concept and explains the growth of this in recent years.[23] In practice, for most commercial and industrial tasks, multi-user systems are necessary, or soon become so.

It has often been remarked that personal computers, and especially those based on the newer 32-bit processors, are more powerful than many minicomputers. In some senses (that of calculating speed and memory management speed) this is certainly true. But in another sense, that of multiple use, this is not so at all. The typical personal computer is not designed to be shared. Only one person at a time can use it,[24] and it is not even designed to be used sequentially, since there is little or no security to protect unauthorized use or accidental destruction of the software or data stored on the computer. The kinds of software available for personal computers reflect this type of individualistic use and in general encourage it.

The gap between single-user and multi-user systems is bridged by networks. A network is a combination of hardware and software, which allows computers to be linked together, allowing for the easy transfer of data between them. Personal computers, minicomputers and mainframes can all be combined in networks in different combinations, allowing for the sharing of information between different parts of an organization using different computer systems. Networks thus provide a way out of the dichotomy between single-user and multi-user systems. Each part of the network can, if necessary, be established as an independent system but with the capacity to exchange data with the other parts.

In practical terms, a form of networking tendency can be seen in the growing emphasis on the linkage of personal computers with mainframes and minicomputers. Workstations of considerable power are replacing the traditional dumb terminals. The personal computer system OS/2 Extended Edition, which is being developed for the IBM PS/2 series of Intel 80386 based microcomputers, will contain specific provision for accessing mainframe databases and downloading selections from them for analysis on the PS/2.

For mainframe and minicomputer operating systems, a significant trend has been the growth in importance of the UNIX operating system. UNIX is notable because it represents a standard that operates over mainframe and minicomputers from different manufacturers. It is also increasingly available for microcomputers as these expand in power and speed. Thus it offers a bridge between all three categories of computer and in principle allows software developed for one type to be usable on all.

What is UNIX? An operating system is difficult to describe in a few words, but it has one important characteristic, its multi-user orientation to which particular attention was paid in its design. Another important feature is its "piping" facility, which conceptualises data processing, input and output in such a way that the user is not concerned with the physical characteristics of devices and can freely and easily change his or her use of them. UNIX is a large operating system because it contains many features that

are optional extras on other systems. It thus spreads standards into areas such as security, data transfer, networks and other areas. The wide-ranging and comprehensive nature of UNIX thus adds to its attractions.

Its critics say, among other things, that it is a cumbersome system which requires a good deal of main memory in the computer to use it, that its syntax is clumsy, that not enough good commercial applications software is available to be run under UNIX, too many different versions of UNIX exist, and the progress in spreading UNIX has been so slow that better operating systems are now available. None of these points are accepted by UNIX supporters, who point out that memory prices have fallen so much that the size of UNIX is no longer an argument against it, that more user friendly interfaces to UNIX are available, and that whereas in the past use of UNIX was concentrated in academic and the scientific community, now much commercial software is available for business applications.

UNIX is not only a technical phenomenon, however, it is an illustration of the kinds of competitive tensions which characterize the world of hardware and software manufacturers. As such, its history is confused, and its future only slightly less so. At one stage it was regarded as a vehicle for challenge by AT&T (its original developers) to the position of IBM.[25] It has also been seen as a means by which many other (mostly European) computer manufacturers could mount a similar challenge, setting up a corporation (X/Open) to do so.[26] Another approach through the traditional standards setting approach seemed to bring consensus on a universal UNIX interface (POSIX),[27] and involved several competing companies. However, a growing commitment on the part of IBM to its own form of UNIX, called AIX, and a developing relationship between AT&T and Sun Microsystems,[28] may have been among the factors leading to the formation of another body, the Corporation for Open Systems, which united IBM with many other companies in a bid to promote open systems based on the X/Open and Posix standards.[29]

The organization X/OPEN is worth mentioning in connexion with UNIX because it shows a remarkable tendency on the part of several hardware manufacturers, particularly European, to combine in what is essentially a market-sharing strategy. The objective is to allow for the free movement of applications software from one make of computer to another.[30] By this means a large common market in software can be created. If successful it would greatly assist the survival of the smaller manufacturers of computers. At present these have to go to considerable efforts to ensure that there is a sufficient choice of software available for their computers. Given the increasing development costs of software and the competition emerging from the microcomputing world, this is an ever more difficult task. For some manufacturers, a commitment to X/OPEN, and to the development of UNIX as a standard, may well have been the only course to follow.

It should be added UNIX is not the only operating system available for different brands and types of computers. Another one, PICK is an operating system with embedded applications, data base management in particular. Traditional distinctions between operating system commands and data base inquiry languages become blurred in such a context. Pick has done quite well in specific commercial fields, but its role in more general applications is as yet unclear.

What does all this mean for the potential software producer? There are two important lessons to be learned. The first is the increasing complexity of operating systems. UNIX exemplifies all this very well. In itself it is complex, with many facilities and many ways of doing things. But as well as

this, there has been over the years an accretion of software which adds to or expands on the basic facilities of the system. For instance, the IBM version, AIX, has added half a million lines of code to the original version of UNIX.[31]

This trend in complexity is also seen in OS/2, the operating system to replace MS-DOS for personal computers using the Intel 80286 or 80386 microprocessor. MS-DOS was supplied on one diskette, while the Extended Edition of OS/2 will need about 16. But it will include considerable database and other capabilities.[32]

Clearly this could make things difficult for the new software house. To master fully the complexities of new operating systems may be beyond the capabilities of a small company. It may not be able to discover all the tricks and hidden short-cuts in the operating system which can be used to give the package a unique character. Again, one of the reasons operating systems are growing more complex is that, as noted, they are incorporating more and more functions previously regarded as extras to be provided as additional systems software or as applications software. Databases and database access software is a particularly striking example of this. The opportunities for third-party software firms are therefore reduced accordingly.

Apart from complexity, however, another clear trend is towards less diversity in operating systems. The move towards a reduced number of operating systems in the market is very clear. The UNIX trend is only part of this picture. In large-scale mainframes IBM has 75-80 per cent of the total market, and has established the MVS and VM operating systems as the de facto standard. In 1986, there were over 20,000 copies installed (which were estimated to generate recurring revenues of $1,5 billion).[33]

In minicomputers de facto standards are less clear with many manufacturers having their own proprietary operating systems. Some forms of UNIX can also be found, offered as an alternative to the manufacturers' own operating system. However, IBM's System 3X computers have been predominant. IBM was reported as giving the system 3X installed base as 220,000 world-wide in 1986. The number of application packages available was given as 4,000.[34]

In microcomputers the dominance of PC/DOS and the similar MS/DOS as operating sytems is overwhelming. The only significant alternative is offered by Apple Inc. The new operating system OS/2, already referred to, will soon make an important impact, even though its final form will probably take some time to appear.

The future horizon, therefore, offers some measure of stability for the new software developer. The trend towards fewer operating systems means larger potential markets. The hardware trends which may ultimately lead to the disappearance of the traditional minicomputer will further reinforce this.

4.3 High-level languages

High-level languages are so called because they allow the user to communicate with the computer in a form closer to that used in normal communication. High-level languages are increasingly important in the software world because of the hardware advances referred to, and because of the increasingly competitive character of the software market. We deal with each of these points in turn. Since hardware change has meant faster processors and memories, it means that it is more practicable to consider using a high-level language in developing a piece of software. Previously, to

get sufficient speed in the final product it was often necessary to use machine code or assembly language. This is in spite of the fact that most programmers prefer to work in high-level languages and it is quicker to develop and test programmes in such languages. High-level languages also have special characteristics which make it less likely that the programme will contain mistakes.

Increased competition in the software market has given a further impetus to this trend, because use of a high-level language can shorten the development time and can give access to a wider market. While a computer maker probably offers a slightly different version of a high-level language, the conversion from one system to the other can be straightforward enough, provided the software developer adheres to some minimal definition of the language and avoids taking advantage of too many of the special features offered in the computer manufacturer's compiler.

In general, the high-level languages in use are very old. Fortran and Cobol have existed in different versions since the 1960s. They maintain a strong user base and formal standardization procedures continue to add new features and improvements to them. The rise of the microcomputer saw a revival of BASIC, first developed also in the 1960s for educational purposes. Business use of BASIC appears to have received an impetus from this. However, there has been a lack of standards and a divergent growth of versions of BASIC far more sophisticated than the original. The spread of new languages is also significant, however. These include C, ADA, and Pascal.

C is the language in which the UNIX operating system was written. It is thus scarcely new anymore, and yet in recent years has grown rapidly in popularity and has moved outside the UNIX community. Its virtues are several: it can attain in many cases the speed of an assembly language while having most of the features of a high-level language. It allows operations which are characteristic of assembly language and machine code, permitting access by the programmer to many aspects of the operating system and hardware devices. This makes it suitable both for systems programming and for process control and other real-time applications. The other advantages of C include a number of sophisticated features such as pointers and data structures (as in Pascal) and list manipulation features (as in Lisp). For all these reasons, the language appears to be being more and more widely used.[35]

Ada is a language developed for the United States Department of Defence, which is now coming into wide use, including outside the military field. The objectives of Ada were to have a single language that could be used for all embedded applications. It would thus replace the use of assembly languages or specialised languages for all real-time applications such as weapons control. All contracts awarded by the Department would stipulate the use of Ada. Once the standard had been defined, software manufacturers began to prepare Ada compilers, the Department of Defence set up a committee to "validate" compilers, i.e. to confirm that they met the required definition of the language, and a large number of compilers for different computers have already been validated. Thus, in contrast to all other high-level languages, Ada's development has been carefully controlled and standardized. The use of Ada is another question: there are indications that because of the very wealth of facilities available to the programmer in Ada, it is perhaps unsuitable or at least unnecessary for many programming tasks. This must mean that, at least in civilian applications, the scope for the use of Ada may be limited. However, interest in Ada and in related development has already extended far outside the United States. In the future, as skills diffuse through the commercial computing community, preferences may emerge for the use of Ada i

the programmers' experience of what is undoubtedly a powerful applications language is a positive one. Accordingly, the conclusion must be that Ada will be an important influence in the commercial field, but that this is still some years away.[36]

Two other languages, Pascal and Modula-2, are both inventions of the Swiss scientist Nikolaus Wirth. Pascal has established a significant user base. It has advantages in many application areas, e.g. both in the commercial and scientific fields, although its success has been greater in the former. An important feature of Pascal has been its adoption by many academics as a more suitable language to illustrate good programming techniques. The theoretical trend towards what is call "structured" programming can be traced to the work of Dijkstra[37], who has laid down basic principles of the discipline. Pascal's advantage lies in the provision of a large number of control structures which allow for the application of these ideas in practice in a way that other popular languages cannot do. By meeting several of the criteria laid down for structured programming it found ready acceptance among those who sought the reliability or verifiability of computer programmes designed according to these principles. Pascal is therefore a case where academic theory did indeed provide a certain impetus for commercial popularity.

Mainframe acceptance of Pascal has been less. Most major computer manufacturers offer a Pascal compiler for their machines, but its use in commercial data processing appears to be limited. The reasons for this are that when change is contemplated (e.g. the replacement of an old system written in COBOL) the designers are likely to be more attracted by the increasingly common C language or by the features of a fourth-generation language, since it may already accommodate features (such as screen handling and a database query language) that are not part of the definition of PASCAL (or of other high-level languages).

Modula-2 is intended to replace Pascal and to remedy its observed defects. Its commercial prospects look mixed, partly because of the wide use now being made of Pascal in those markets which might have used it. However, its longer-term influence on language definition and the next generation of computer languages may be considerable: the true modularity possible in it, and the power of the user to define communications between different modules, are features which are likely to be seen as very desirable in any future definition of the correct functioning of a computer language.

4.4 Fourth Generation Languages (4GLs)

These are proposed by their producers as a replacement for the traditional data processing languages (such as Cobol, RPG, PL/1, etc.) which are regarded as third generation. (The second generation would include assembler type languages which provide mnemonic and limited macro facilities, i.e. one step removed from the first generation, the machine code used by the first computers).

Fourth generation languages have to be distinguished from languages such as Ada, which provide a number of new concepts for practical use by programmers. 4GLs are intended to reduce programming: the statements and commands encapsulate many lines of Cobol or PL/1 code. Operations that might take a large amount of space and time to specify in a third-generation language can be conveyed in one or two lines, which are often in a form that is nearer to English.

What might be called a subset of 4GLS is the programme or application generator. This is essentially a compiler whose output is a Cobol program. Give the set of commands which it is desired to implement, the program generator then produces appropriate Cobol statements. The advantage of this is that the code is portable (in principle) to another Cobol environment. Another advantage is that the code produced can be examined and directly modified in a way that is not possible with a normal compiler whose output is machine code.

A number of features may be part of the 4GL: screen handling, data entry, printing formats, database management, access and updating. All these aspects, if they are included, will be distinctive. Standards, whether formal or de facto do not exist at all in this area. Experience gained in one 4GL may not be of much use in another. Here is a field, however, where competition is increasingly fierce between different groups. The computer manufacturers now often offer some form of 4GL, but the large software companies are also extensively involved.

The future for 4GLs is promising. The driving force is the shortage of programmers and the limited increases in their productivity, combined with increasing pressures to computerise as many aspects of commercial life as possible. Competition combined with the increasing complexity and changing character of commercial data to force companies to improve and expand their information systems. However, the typical effect of such pressures is a growing backlog in the data processing department of a company. 4GLs appear to offer a fairly easy way out of this difficulty, since their main claim is that applications can be developed far more quickly than by traditional methods. The drawback is that they require a certain commitment of resources, a certain amount of retraining of staff, and, most importantly an abandoning in many cases of patterns of work which have been its use for perhaps twenty years or so. In addition, the very pressures on data processing managers will tend to limit their scope of action. Concerned with maintaining, modifying or patching up existing systems they will not have time properly to examine the alternative approach to problems offered by the 4GL.

The other highly significant feature of 4GLs is their convergence with database management systems (DBMS). The 4GL may offer a DBMS of some kind as an integral part of the language or "environment" (a somewhat vague term used to describe anything from a user interface to a suite of programmes marketed by the same producer with some commonality of command structure and syntax). Alternatively, the DBMS may offer sufficient facilities in terms of data input and output and report writing, as well as data storage, so that its command structure is in effect a 4GL.

A good illustration of the convergence is the spread of the SQL language. This is used to access data from relational databases. Developed by IBM, it has now become part of the X/Open standardization effort as well as being defined in an ANSI standard. It is now available for many DMBS systems on mainframes, minicomputers and personal computers also.

4.5 Database Management Systems (DBMS)

The growth of the DBMS, and just as significantly, the broadening of its definition may be seen as a process where software has developed to meet the practical revealed needs of computer users. The third generation language Fortran grew out of the need to express scientific relationships in familiar, formula terms. Cobol was developed to the order of the United States Department of Defence, but it grew again out of the need to express typical commercial calculations.

Third generation languages deal with atomistic data. They do not have built-in methods of handling lists, or collections of data of different kinds such as "name, rank and serial number". Equally important, the storage as such is not defined. The languages are intended to process data i.e. to read it into the computer, do something to it, and then write out the result, whether to a screen, to a disk drive or to a printer. Even on-line systems can be just a straightforward extension of this particular concept.

The DBMS provides a way to store data in such a way that some (or in some cases all) of the relationships within the data are preserved and can be followed through in searching it. It also provides a way in which the data can be selected according to different criteria and summarized. Typically this summarization procedure will be in the form of printed tables of results. In this case the stage is called "report generation". The DBMS may also provide a wide number of other facilities. These facilities can include a number of ways of dealing with the input of data, including its specification with respect to form and content in such a way that only data of the correct type can be entered, and only in certain circumstances and by persons authorized to add or amend data in the database. The facilities can also include a good deal of "housekeeping" functions invisible to the user or the constructor of the database but nevertheless important in ensuring that the storage media (for example, hard disks) are used in the most efficient and economical way.

DBMSs can be classified in many ways, and they range from systems essentially analogous to card-files of the conventional kind to other systems of an elegance and complexity sufficient to exceed many ordinary commercial needs. A common classification scheme is to divide DBMSs into: three categories: hierarchical, network, and relational. It is the last category which is the focus of most new development and competition.

Relational databases are so called because they store the data in such a way that connexions can always be traced and revealed as required. The origins of relational DBMSs can be traced directly to the famous article by Codd, which set out criteria for the establishment of the relational principle and pointed out that no existing system met these requirements.[38]

The basic idea of a relational database is that it treats all relationships as composed of pairwise relationships. This allows all data to be stored in the form of two-dimensional matrices (i.e. tables). Depending on the data, it can therefore appear in more than one table. At first sight this duplication (or indeed multiplication) of storage requirements appears very wasteful. However, the design of the DBMS itself can minimize any duplication, and in any case with the rapidly falling costs of hardware this criticism will tend to be less important.[39]

In spite of the difficulties associated with transferring existing systems, relational DBMs are now making a considerable impact. The main reason for this is not so much the perceived virtues of relational technology but the pressure for DBMS in general. The information within an organization is increasingly seen as its most important asset, and easy and flexible access to it is increasingly required. The role of dataprocessing departments is tending more towards the construction and maintenance of a database and the control and facilitation of access to it. Thus, dataprocessing becomes a continuous task rather than the series of discrete "jobs" into which it used to be decomposed.

4.6 Computer Aided Design (CAD)

Discussion of CAD is often bracketed with that of computer aided manufacturing (CAM) although in practice it is usually CAD that is being talked about when CAD/CAM is under review. This is because of the role of the computer in automating the process of design, a fairly clearly delimited sphere. Roughly speaking CAD/CAM means in practice the automation of the design process in manufacturing. When CAD is combined with a wide range of diverse informatics techniques such as robotics, remote sensing, process control, numerically controlled machinery, automated stock handling, ordering and inventory management then, provided all these processes are linked in a common communication and control system, the whole approach is known as computer-integrated manufacturing (CIM).[40]

How can design be automated? Strictly speaking it is not the creativity at the heart of the design process that is automated but rather a number of mechanical tasks which the designer has to carry out in order to realise his or her objective: a set of drawings and instructions for tooling up for the construction of the required object. The designer draws on the screen of a computer terminal. When the result is satisfactory it can serve as the direct input to production (e.g. by controlling the movements of a machine that cuts or shapes some material according to the design that has been achieved). More simply, the design can be produced as a drawing to be used by the production designer.

Drawing on the screen is accomplished by allowing graphical input from the designer and using a number of software and hardware tools. The graphical input can take several forms: a digitised image of a handdrawn sketch can be used, or a joystick, or a digitizing tablet, a mouse, or cursor keys. These treat each point in the drawing as a series of co-ordinates to be afterwards manipulated, the rough initial sketch being refined on the screen to a finished design. Facilities for doing this vary with the sophistication of the hardware/ software, but include for instance, the automatic joining of points (line drawing), enlargement along a specified axis, rotation about a specified axis, etc. The generation of three-dimensional objects (viewed, of course, in two dimensions on the screen) is also included, (with a selection of basic shapes provided as building blocks for the image). This allows an object to be examined from any desired point of view, and can be a means, for instance, of estimating the actual appearance of an experimental design, and thus replacing the construction of solid models as may have been common practice in many industries. A cup, a table, a box may, as a computerized image be modified and re-examined and appraised in a way that allows he designer rapidly to assess the acceptability in production or consumer terms of a proposed desgin. (Another example from architecture is a package which allows the user to design a building and "walk" through it: given the structure of the proposed building the computer can calculate how it would look not just from outside but from any position within any room of the building also).

It is not perhaps surprising that some of the most sophisticated applications of CAD, in fact true CAD/CAM applications, are found in the electronics field. The design of electronic circuits on a screen, with such features as automatic minimization of interconnections, is now standard. The output can range in sophistication from a printed design, to data in a standard format, to specific control sequences for the machines which manufacture the chip.[41] Even in the highly specialized field however, there has been a trend towards standardization: a growing commitment to, for instance, the IGES, MAP and OSI standards, together with such de facto

standards as the Gerber data format and the Sun workstation as a "platform". The growth of technical workstations has meant the increased availability of large sophisticated processing power in individual units: it thus expands the potential market, in unit terms, to a considerable degree. This has provided an important stimulus for the creation of software companies in the design field.[42]

4.7 The impact of artificial intelligence techniques

4.7.1 Overview

The development of artificial intelligence (A.I.) technique is already having important effects on the world software industry. While debate continues at a theoretical level as to what is artificial intelligence and as to whether there are limits to its future development, it has as its main concern making computers replace some primary human function such as sight, speech or analytical thought. An alternative definition is the following:

"A.I. is concerned with programming computers to perform tasks that are presently (sic) done better by humans, because they involve such higher mental processes as perceptual learning, memory organization and judgemental reasoning."[43]

A.I. Techniques are influencing the following areas in particular:

- speech input and output
- pattern recognition
- expert systems

Why is A.I. becoming increasingly important? The first reason is that the enhanced capability of computers makes possible the application of many A.I. techniques which rely on a good deal of processing, typically in the form of searching for a matching pattern between input and some stored database. In this sense the software development is driven by developments in hardware. There is also a growing demand for applications in speech and character recognition, where the object is to automate the process of data entry. (This trend is of course encouraged by the growing availability of enormous mass storage in the form of optical disks).

However, a definite if difficult to measure factor influencing the growth of the A.I. market is the push from software producers. Many of these are casting round for new product ideas. The personal computer market is saturated with spreadsheets, wordprocessing packages, and integrated packages of various kinds. The mainframe market still has many possibilities for increased sales in the areas of DBMs and 4GLs in particular, but this market is weighted heavily in favour of the computer manufacturers, and in favour of larger companies who can field a sales force and provide service and support to a fairly conservative market. Thus, the search for new product ideas enters the area of A.I. This is not, of course, the only area in which activity in the development of new software products is taking place. However it appears to many as the area which offers particular scope for innovation and promises great rewards for success.

Other factors influencing the growth of A.I. include advances made on the theoretical side by researchers especially in universities who find the subject and the problems interesting. Specialized demands, such as those by international organizations for automated translation and interpretation, or the need to deal in data processing terms with a representationally complex language such as Chinese also combine to give a further impetus to research in this subject.[44]

Robotics and the automation of the manufacturing process are another cause of the growing interest in A.I. applications. The constant search for an improved competitive position and, especially in developed countries, the need to minimise total labour inputs to the production process have caused accelerating interest and applications in this field. (Other factors at work in the spread of automation of the production process include the need to minimise use of materials through more efficient cutting, measuring, etc. the need to speed up delivery times, the need to meet a wide variety of demands for differentiated products, and the need to meet rapid changing demands or changes in the relative costs of inputs).

Robotics and automation are not per se a branch of A.I. But they increasingly use A.I. techniques, especially in such areas as sensing. This means that machines are made to recognize objects or characteristics of objects, either through direct contact or by vision, where a digitized television image of the object is captured and tested as to whether it matched a pattern already known to the machine, and action is then taken accordingly.[45]

Automation also includes the speeding up of still manual tasks of control, such as can be by speech input and output. For instance, instead of having gauges and indicators, a machine could "speak" the temperature, pressure or whatever. In difficult industrial conditions this may greatly improve the carrying out of specialized tasks, since the operator is not distracted from them by having to monitor a dial, a graph or a digital read-out. Similarly, speech input can allow control of a machine or a process without the operator having physically to touch a button or a switch. Even simple processes such as stocktaking can benefit from speech input, since the stock-taker can simply call out the numbers and types of the items being counted, rather than entering them at a keyboard, using barcode readers or any other direct handling of the objects or equipment.

4.7.2 Expert systems

Expert systems are the branch of A.I. applications in most widespread use. They are systems which attempt to embody human knowledge and expertise in an accessible form, typically leading the user through a series of questions to resolve a problem.[46] Expert systems are now a commercial reality. All the main computer manufacturers offer products in this field. The pioneering academic centres of expert system work, such as Carnegie-Meller, Stanford, Edinburgh and Marseilles continue to lead the university world in the subject, but the bulk of activity (often hidden) is in the commercial field.

In hardware, it has been noted that many purchasers are turning away from dedicated artificial intelligence workstations, and instead increasingly taking the much cheaper option of buying a software package to run on an existing personal computer such as an IBM PC or equivalent.[47]

In software the proliferation of expert system shells continues. A shell is the software needed to construct an expert system, and written in a generalized form so that it can in principle be used for any subject area. Shells are offered for sale at prices from several hundred dollars upwards, and new ones almost invariably for microcomputers rather than for minicomputers or mainframes. However, they vary very much in quality, sophistication, and ease of use. Users also continue to develop their own expert system software. For this purpose the Prolog programming language has made considerable progress. However, the United States dominance in the

commercial expert systems field means that the LISP language still is the major one, since a heavy investment in this language has been made by software developers and computer and workstation manufacturers.[48]

The use of expert system shells will continue to increase, but it is likely that a plateau will soon be reached. This is because some disappointments can be expected: the naive business user who purchases an expert system shell to be used on a personal computer will gradually realise that he has bought merely a tool, and that the task of constructing a useful expert system is a fairly long and demanding task that still lies ahead of him.[49] New shell products can in the immediate future offer improvements only in fields where progress has from some points of view been already sufficient to be going on with e.g. in faster inferencing, in more detailed explanation of the processes by which decision were arrived at, in the incorporation of more user-friendly techniques, etc. Progress can also be expected in the area of constructing the knowledge base, allowing more flexibility in the specification of rules, for instance. But the real problems of expert system construction remain, strictly speaking, outside the realm of software. The decisions as to how to limit the problem area, how to ask the right questions of the human expert and how to decide what are the important parts of his answers are difficulties which the present levels of software development cannot easily deal with and from some points of view have little if anything to offer.

For this reason a strategic choice for development in this areas would be better in the areas of construction of the knowledge base rather than in the design and implementation of another expert system shell. The construction of knowledge engineering tools to detect inconsistencies at construction time, to suggest gaps in the information supplied, to maintain checklists of points to be covered, and to detect and suggest promising lines of enquiry in the interrogation of human expert are some of the types of software that could be developed. To enter such a field of work for a software manufacturer may be a promising route for some developing countries. Since experience is still limited in both developed and developing countries, it any not be as difficult to compete internationally as in some other fields. The capital equipment costs are not high, even though there will probably be a foreign exchange component.

5. GENERAL PRODUCTION STRATEGY

5.1 The stages of software production

Traditionally, software production has been described mainly in terms of the design and implementation of computer systems for use within an organization or a complex piece of equipment. Much attention has been given to stages of such an activity. A typical classification is the following "software life cycle":[50]

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- System testing
- Operation and maintenance

From this and similar points of view, the production of software is something which extends well beyond the actual design and writing of a programme. It calls for a number of diverse skills including human communication, personnel management, resources planning, etc., as well as the traditional skills of the systems analyst or programmer. Taken together in an integrated way, the considered application of these selected skills has become the discipline called "software engineering". The rise of the discipline is a response to the growing complexity of the task. As a consequence of this development, considerable examination has taken place of the various stages of the development of a system, with rules and methodologies which can be applied to the practicalities of software production. The allied question of the costs of software development has also been explored,[51] especially in view of the kinds of complex systems under development, often involving large teams of staff and sometimes at different locations.

However, from the point of view of the software entrepreneur, the process has not been as well analysed. The conventional analysis assumes that the idea already exists of what the software is to do. It assumes also that, on completion, the system will be used, perhaps after modification in the light of the intended user's experience of the first version. But the software entrepreneur faces two other stages: one is the initial one of deciding what to produce, and the other is the later stage of trying to sell it when completed. The task is essentially a speculative one. The intermediate stages, of design and implementation, have some similarity of characteristics whether the software is for internal use, is being prepared to order, or is to be marketed as a package.

5.2 Product ideas

Ideas for new products come from an understanding of existing software and perhaps also an understanding of another specialized area. The potential software producer can be someone of a technical or professional background whose idea for software derives from an understanding of the way in which a computer can help in the work. Equally, if the potential producer has access perhaps through preparing custom software or providing computer services to some specialized company or institution, this may also provide ideas for new products.

We can make a rough classification of ideas for new products as follows:

- Invention
- Replacement
- Synthesis
- Link-type
- Captive
- Local
- Embedded
- Custom

The first type, Invention, refers to the genuinely new idea for software, something that has not been tried before. While in the early days of computing much new software could have been put in this class, it is no longer easy to come up with ideas which are genuinely original and at the same time practicable. A good example is the spreadsheet programme. The first of these, VisiCalc was certainly an invention. The hardware conditions for its invention were present in that it is a highly interactive concept: only the availability of personal computers allowed it to be successful.[52]

The second type, Replacement, includes the presentation of existing ideas in a new form. The improvement can relate to a number of different areas, such as the speed of calculation. This is particularly important in database software, were a select or soft can, if complex, consume considerable resources. Speed is also important for compilers and for CAD software.[53] Ease of use is another field where improvements can readily be made to an existing idea. This is encouraged by developments on the hardware side referred to earlier, where the user interface is enhanced by high-resolution screens which can show features such as menus, icons, windows, etc., together with mouse pointers, and speech input and output. An otherwise conventional package can distinguish itself and gain a competitive advantage by making full use of available features of the hardware and software. Ease of use is, of course, a wider concept. It can include the provision of extra functions within a new package to carry out operations which experience has shown are necessary but tedious to perform with the existing package. Improvement can also be in the area of removing restrictions on the size of the problem to be handled by the software, these restrictions existing because of limited imagination on the part of the original developer or hardware restrictions which have been made irrelevant by new developments.

The third type, Synthesis, refers to the creation of a new class of software products by the merging of the functions of separate classes. The best example is the integrated package which can include most of the popular office software functions word processing, spreadsheet, database, graphics, communications, etc. As noted earlier, integration of this kind is a continuing trend, brought about very often by the increased availability of computer memory. Very often, the process of synthesis exploits the innovative character of earlier products. What was a new idea, and was sold separately, becomes a feature of the later package. There is, however, a sense in which the combination of existing ideas also represents a new idea. For this reason the search for new product ideas has to include consideration of existing ones, with special attention being paid to the possible merging of ideas from less obviously linked areas.

Link-type products represent a promising field for software development, but it is one which changes very quickly. By a link-type product is meant one which overcomes improved communication between hardware and/or software systems. The definition thus includes software to carry out such tasks as: converting a file from one format to another, or translating some code from one language to another. It could also include software to allow, for

instance, a computer to use a printer which had not been designed for this type of computer. At first sight, the possibilities for this sort of product might be thought to be limited in the longer term by the trend towards standardization, but this is not so. Firstly, universal standards are still years away, secondly even the best standard is made obsolete technological change, and thirdly the very push towards standards creates a need for software products which allow non-standard hardware and software to communicate.

Captive software is the name used to describe software products such as "add-ons", which are software providing extra features or enhancements to successful products. It can include also other products which are intended to improve the performance of successful products, or, indeed, are intended to monitor the performance of complex software in a way which will allow improvements in its use.[54/] All these share a common characteristic, that they are depending on the success of another software product. Their potential market size is determined by the number of copies sold of the other product. More importantly, it is very vulnerable. For instance, the producers of the original product may decide to bring out a new version which incorporates most of the features of the add-on products. Again, a new version may be incompatible with the existing version of the add-on. This may not be intentional on the part of the producers of the main software but it can nevertheless create considerable difficulties for those whose products are supposed to work with it. For this reason such software products can be called captive: they are wholly dependent on the success of another. In that sense they are analogous to software products which are almost always either hardware or operating system dependent, or both, but the difference lies in their limited functionality: what they do for the user can be quite restricted, and the need for the function as well as the product can be eliminated at any time by developments outside their control.

Local software means the meeting of a particular national practice by appropriate adaptation of existing software or software ideas. This can amount merely to changes in the language of the command structure or user interface of a package. If, however, the language is very different there will have to be more elaborate changes. Local business or governmental procedures in the form of accounting, taxation, etc. represent the most common origins of local software. Although local software is usually produced in the country concerned for that market (or else carried out for instance at the European headquarters of a developed country software package producer). But it does not have to be so, of course: a developing country software company who produces a standard accounting package to meet the special needs of the country has mastered many of the skills needed to do the same for another developing country also.

Embedded software is that contained in a machine vehicle, or piece of equipment. As such it is usually developed by or on behalf of the manufacturer, and is similar to the final category, custom software in that the aspiring software entrepreneur has to produce this product to order and is essentially a sub-contractor. While the product itself is not marketed by the software producer there is nevertheless, especially for the new producer, a good deal of marketing involved, since it is necessary to convince the client that the software company has the skills and experience to carry out the task correctly and on time. The difference between embedded software and custom software is partly in the markets: manufacturers of capital goods and consumer durables are the typical users of embedded systems, while custom software could be written for anyone. Embedded software is used by microprocessor or microcontrollers built into the equipment. It is usually

very time-dependent and is usually written in assembler or in a special control language (increasingly, Ada is being used). Embedded systems construction may also in many cases require a much more detailed knowledge of hardware considerations.

5.3 Choice of market

Clearly, the possible market will be partly determined by the idea for the product. The idea may be in practice specific to a particular application sector, a particular country or a particular type of computer or operating system. To have as large a potential market as possible is of course desirable, but in the case of software production it is particularly so because of the low marginal costs involved in expanded production.

In terms of large potential markets, consideration must be given not only to the installed base (the number of relevant computers in use) but the type (mainframe, mini or personal computer). The price at which the product can be sold varies accordingly, universely with the number installed: thus personal computer software, unless highly specialized, can hardly be sold for more than about US$ 600. Mini and mainframe software is considerably more expensive, and in fact is often rented, thus giving the producer a certain control over future revenues.[55/] The question of pricing in general is also affected by technological change: the shortening of product life-cycles in the software field means that pay-back periods have also to be shorter if they are to exist at all.

A full understanding of the selected market is necessary and in particular of the motivations of the potential software purchasers, which vary widely depending on the context in which their computing work is carried out. For instance, the IBM mainframe installations constitute a large and stable market (with respect to minicomputers, a similar judgement can be made). However, the mainframe installations also constitute a market which:

- is concerned as much or even more with quality and service as with price;
- has already been targeted by many others;
- is in fact fragmented, since so many software products have become "standards" within this typical environment. It is not enough to be compatible with MVS. There is other system software (in teleprocessing, for instance) which has to be taken into account. Sophisticated heavily marketed products such as database management systems have specialised formats and imply certain ways of working. Can the new product fit in with these?
- is an evolutionary market, not very much open to significant departures from established ways of doing things. This is principally because of the need to keep established computer systems operating. These have often entailed considerable investment costs and they provide information flows which cannot be interrupted. For this reason any new product has either to complement existing systems or else provide a feasible way to replace the existing system in an orderly manner. As noted earlier the computer manufacturers have to provide a stable and secure environment for the purchaser. The user has to feel that the equipment bought is not likely to become obsolete quickly. In fact, as we have seen, it usually does become obsolete quickly, in the sense that something both better and cheaper will appear on the market soon after the purchase has been made. But it can be said that the purchasers grudgingly accept this, if their individual expectations are met, if the available equipment is within

their budget, if it will do the job it is intended for, and, most importantly if it can either be physically upgraded (which is better) or else will use the kinds of software that can also be used on the next generation of the computer in question.

This last requirement means that the computer manufacturer has to provide an "upgrade path". This can take several forms, but the simplest is a set of programmes that convert software written for the old computer to software that will run on the new. Such a conversion is not always necessary since the hardware may have been designed to be "upward compatible". However, on other occasions it may be necessary to abandon such compatibility in order to exploit fully the possibilities of the new hardware design or components. It is in these cases that the provision of an upgrade path is needed. Such a path can also include the provision of hardware modifications, and the prevalence and cheapness of integrated circuits in many items of equipment makes this a relatively simple task, since the replacement of one IC by another is a relatively simple matter. However the provision of software that converts old systems to allow them to operate in the new environment is one of the most usual ways in which an upgrade path is made available.

There are cases, however, in which an upgrade path is not provided by the manufacturer. This can result from the purely practical calculation that the returns will not justify the development effort, but it can also be that the benefits of the new technology are sufficiently obvious and significant for the user to simply abandon his old systems, running them in parallel with the new ones only until the latter have proved their reliability.

In other cases there may be a "generic" shift in the technology. An example is the movement from 8-bit to 16-bit microprocessors, where the earlier generation (8080, Z80 and 6800) was not compatible with the later (8086, Z8000, 68000) in terms of the instructions understood. individual manufacturers may have provided upgrade paths for their own series of microprocessors, but many microcomputer manufacturers took the opportunity of a technological shift to change their sources of microprocessors. Third-party software companies, as well as chip manufacturers, offered products, cross-assemblers, which allowed the instructions of one microprocessor to be translated into those of another. Thus, technological change created a gap which was filled by third-parties. Many other gaps necessarily arise, as long as the technology continues to develop, and such gaps always represent opportunities for third-party manufacturers to provide a product which can fill the gap and allow two previously incompatible pieces of equipment to communicate with one another.

In considering markets, therefore, it is important to reiterate that technological change creates markets in two distinct ways. Firstly, through direct progress and the provision of new hardware it encourages new ways of doing things and, accordingly, new software requirements, for software products which exploit the possibilities of the new hardware. But there is a second type of market, one created by the inconsistencies and discontinuties of technological change. Progress continues in many directions, and led by many different companies in several countries. To reap the benefits of more than one advance simultaneously may mean for a new product developer, a difficult problem of integration: how to fit together a sensor, a processor and a memory management unit in such a way that it will be of interest to users of another product that happens to be highly popular but with which no suitable interface exists. Integration, harmonization, communication are all goals of the movement towards hardware and software standards. But in the absence of uniform acceptance of standards, and in the face of the continued

assault on standards represented by accelerating technological change, the need persists for products to fill the gaps between the advances on different fronts. The link-type product idea is only one manifestation of technological change, which increasingly determines the whole market environment.[56]

It was pointed out above that simply to follow what appeared to be the largest or "standard" configuration of equipment may not necessarily be the best approach. On the other hand to follow a more specialized market carries its own risks. One of these is that the market may be about to be eroded, or may be wiped out by the emergence of some new technology or a rival product. Here the development time is crucial in a way that is not necessarily true for products directed towards a more stable or conventional market. In the latter case, a product may still find buyers whether it appears in twelve months time or eighteen months time. In a more specialised market, it may contract rapidly as the technology changes and the total number of potential purchasers can diminish rapidly. This is still the case even if the number of a particular computer in use, for instance, remains the same. Although none are immediately abandoned on the announcement of the availability of some superior product, decisions are nevertheless taken on future purchase. Once such decisions are made, it is then only for very compelling reasons that any software product would be bought for the existing computer. Even though it may remain in use for another year or even more, the software market is disappearing rapidly.

Being too near the frontier of hardware developments is equally risky, since the new development may fail to attract a wide market and the work that goes into building software for it, together with the consequent investment, may be lost. However, many promising niches exist at this frontier. New developments in the mainstream of hardware are also important and here the established manufacturer of software is often at an advantage over new entrants. This occurs particularly, for instance, in the case of microcomputers, whose manufacturers are anxious to launch the machine on the market with as wide a software base (i.e. selection) as possible. This means that they will make available to software companies a prototype version of the new computer, or at least disclose to them the details of the hardware and operating system so as to allow the software manufacturer both to have a product ready for market launch simultaneously with the launch of the computer itself, and also to enhance the attractiveness of the computer to potential purchasers through the provision of software that exploits any special feature of the new hardware design. The companies selected for this favoured treatment are typically those who have produced successful packages in the past, since too wide a distribution would discourage some of the software manufacturers who would feel that their competitive advantage was being lost, and might also result in disclosure of any still secret aspects of the hardware design.

Such a practice makes it more difficult for new entrants to compete. It favours existing successful software companies. But the practice was not developed for the sake of these companies, but rather by the hardware manufacturers to ensure success for their products. Therefore, they remain to some extent open to broadening their range of favoured software companies, provided they are satisfied that the company entrusted with details of their forthcoming product is capable of producing something worthwhile in the way of software for it. If can therefore become very important to cultivate selected hardware manufacturers, to demonstrate the software capabilities of the new company and to convince the hardware developers that the connexion is worth developing. Such a task may not be an easy one: it can require in some cases a physical presence in the same country as the hardware manufacturer or at

least very frequent visits to it, and for a small company in a developing country this may be an overambitious undertaking, given the foreign exchange costs of establishing a presence in North America or Western Europe.

However, some such contacts are essential for any software production strategy targetted towards the leading edge of technology. To be familiar enough with present and future developments to be able to identify an emerging software product opportunity is not easy when the research and development activity is far away and sometimes hidden for commercial reasons. But there are many methods of monitoring technological progress in this field which are relatively inexpensive. Journals in electronics and computing are often inexpansive because of the high quantities of advertising that they carry. professional societies are usually very keen to expand their international links and often have a proliferation of specialized interest groups which act as a clearing house for information (and often standards) on particular hardware or software topics. Again, hardware manufacturers often provide a great deal of printed material on their own products or on technologies which they use. This material is often free or else costs very little. Attendence at large trade fairs, such as CeBit or Comdex, is a relatively easy way of becoming familiar with a complex and rapidly changing market.[57]

Co-operative arrangements for exchange of scientific and technological information are found both within the United Nations system and as part of other inter-governmental arrangements. UNIDO has actively promoted information exchange both through the TIES system and more specifically in the field of informatics technologies through schemes such as the REMLAC regional microelectronics network for Latin America. The fledgling software company can do worse than investigate official channels in its own country, to discover what information systems are accessible as a consequence of international agreements.

The question of software protection should also be mentioned. The subject is a complex one since it includes technical as well as legal considerations. The investment made by the producer has to be protected through some system or systems which will prevent unauthorized copying of the software. This can lead to a loss of potential revenues, and means to guard against it range from physical protection of the software diskette to the application of patent and copyright law.[58]

5.4 Investment costs

The investment costs associated with software production include the provision of hardware and software equipment. Clearly there will be a need for computer hardware, the same as or better than that for which the software will be written, although development hardware which can simulate the largest hardware may also be a possibility in a few cases. Inefficient hardware or limited access to it does not make it impossible to produce marketable software but it adds to the development time. This might not appear to be so much of a problem if local personnel costs are low, but if the product under development is in any way sensitive to technological change, then development time can be decisive in determining the success of the product. Too long a wait may make it obsolete, or the problem it is to solve has been taken care of by someone else.

Investment in software is at least as important for this reason, and for others. The choice of the correct tools will not only reduce the development line but will also make the final product more attractive and reliable. Software products are available which are designed to assist in all stages of

the production of software, including design, production, documentation and maintenance. The product includes the following:[59]

- Design tools, which allow the conceptual structure of the software system to be analyzed;
- Optimizing compilers, which produce efficient object codes by analyzing the source code for redundancies;
- 4GLs and application generators, which may also have a number of other tools as features;
- Screen generators;
- Many forms of debuggers, which check the source code for errors;
- Execution flow summarizers;
- File comparators;
- Translators.

Many of the features found in such software tools are also to be found in the new generation of software tools known as CASE (Computer Aided Software Engineering). These include what are called programmers workbenches or analyst workbenches. The more comprehensive type is called an Integrated Projects Support Environment (IPSE), capable of controlling large software projects and ensuring consistency in the work of the different people involved and generating comprehensive documentation for the product.

These products vary in capabilities and in price. Their potential contribution to the success of the final product can, however, be very great, and productivity improvements of 20 to 30 per cent have been cited of IPSEs.[60] The cost of such tools may therefore be a centrally significant feature of any detailed feasibility study for a software product, and the savings in labour inputs and project time, as noted, must be taken into account. This is so although estimating the work involved is difficult. Most cost estimate methods for software production require the number of lines of code as a starting point.[61] It can also be important to follow design procedures which allow generalized parts of the software to be used again in subsequent products ("re-usable code").

The second main advantage of such tools, that they greatly reduce the potential for errors in the final product, will have an impact on the cost structure at a lager stage. A faulty product will not only give the new producers an unfavourable reputation[62], it may also lead in some cases to liability for consequential damages. As a result, there has been a growth in independent companies which for a fee will test a software product for errors before it is marketed. Their customers include very large and successful software producers.[63]

FOOTNOTES

1/ At the level of machine language, however, this distinction can become rather tenuous.

2/ OECD (1985a), p. 20.

3/ This is in fact the case with the IBM Personal Computer System/2 and Apple Macintosh computers, both of which use the same size diskette (3.5").

4/ See United Nations (1968) and United Nations (1975).

5/ Software is sometimes specifically referred to as an "intangible (fixed) asset" in taxation regulations. See OECD (1986).

6/ But not necessarily in familiar ways: the construction of "macros" in spreadsheet applications, and even the storing of key sequences in a function key is a kind of programming and thus amounts to software creation.

7/ Schwarz (1987) cites the example of the Burmese Post and Telecommunications Corporation, which acquired payroll software that allowed for far fewer standard deductions than are usual in Burma.

8/ See Arossa (1988).

9/ United States Department of Commerce (1986).

10/ See Boehm (1987).

11/ "Survey: Software Packages for Business", Financial Times, 1 May 1985.

12/ "Survey: Computer Services", Financial Times, 15 October 1987.

13/ See also OECD (1987). For an illustration of the official standards development process, see United Nations Economic Commission for Europe (1987), p. 96.

14/ See Jackson, P., "Add-on flood keeps the tide flowing into Lotus channels". Computer Weekly, 9 April 1987. Many successful commercial software developments can also lead to official standards subsequently, such as IBM's database access language SQL, which is the subject of a proposed ANSI standard.

15/ For detailed descriptions of some RISC processors, see Gimarc and Milutinovic (1987). However, the article does not cover important new RISC processors such as the Sun SPARC and the Motorola 8800.

16/ In fact, microprocessors do not always fit neatly into one of these categories. The Intel 8088 was the heart of the original IBM Personal Computer: it had a 16-bit calculating capability but an 8-bit data bus.

17/ See, for instance "Forget the hype, this is the real thing", Computer Weekly, April 23, 1987. (Superconductors will in the longer term have significant effects on the energy field also: see Official Journal of the European Communities, No. C 46/19, 18 February 1988, pp. 19-20).

18/ Bernhard (1985). See also Labich (1988) for some commercial issues in parallel supercomputing and the role of IBM and Cray. On some differing views on the future of parallel processing see "Titans engage in battle over tomorrow's systems", Computer Weekly, 26 May 1988.

19/ For a simple summary, see Weber (1988).

20/ For an overview, see Conklin (1987). See also Nelson (1988).

21/ A 14 inch WORM disk holds 6.8 gigabytes (i.e. 6800 megabytes) and typically costs $750 (1987). The 5.25 inch size, holding 0.8 gigabytes, is becoming popular in "jukebox" systems. It costs $125. CD-ROM unit costs are much lower, but there is a fixed cost of about $15,000. See IMC Journal, Vol. 24, No.4, 1988. However, mastering costs as low as $1,500 and duplication costs of $2 have now been reported ("Nanobytes", Byte, August 1988).

22/ See United Nations Economic Commission for Europe (1987).

23/ The movement from a "Ptolemaic" to a "Copernican" concept of data processing is described in UNIDO (1988). For a general introduction to database ideas see Martin (1981).

24/ Multi-user operating systems are available for many models of personal/microcomputers such as XENIX (a form of UNIX) from Microsoft Corporation. However, the overwhelming majority of such computers have single-user operating systems.

25/ See "A tarnished Unix fails to lift AT&T", Computer Weekly, 16 July 1987.

26/ See Isaak (1986), reported in Data Processing Digest, October 1986. See also "Group of ten challenges IBM", New Scientist, 27 November 1986. The first step has been the preparation of the X/Open Portability Guide, describing the Common Applications Environemnt (CAE).

27/ Posix is a standard for interfacing with UNIX, set by the Institute of Electrical and Electronic Engineers (IEEE 100 3.1). See "Just wild about Posix", Computer Weekly, 29 January 1987.

28/ See "Software: Why AT&T and Sun hooked up", Electronics, 29 October 1987.

29/ The Software Foundation is supported by Apollo Computer, Group Bull, DEC, Hewlett Packard, IBM, Nixdorf, and Siemens. See IEEE Computer, July 1988, p. 62. Its work will be based on the IBM version of UNIX, called AIX.

30/ X/Open's work of standardization is concerned not only with UNIX but with applications languages, so that an application may be written in a certain form of COBOL or FORTRAN and then run on any X/Open standard computer without the source code having to be changed. X/Open is including SQL (see section 4.4) in the next version of CAE.

31/ IBM advertisment, Computer Weekly, 26 November 1987.

32/ See Malloy, R. (1988).

33/ See Fertig, R., "IBM sets its sight on greater software revenues", Computer Weekly, 27 March 1986.

34/ See Fertig, R., "What IBM has in store for users", Computer Weekly, 11 June 1987.

35/ For a review of the C language, see the articles in BYTE, August 1988 or Kernighan and Ritchie (1978).

36/ See Pyle (1981).

37/ See Dahl, Dijkstra and Hoare (1972) and Mille (1986).

38/ These criteria were set out in Codd (1970). More recently, Codd is reported as having encreased the number of rules to 166 features grouped in 13 classes. (See Computer Weekly, 8 October 1987).

39/ Except insofar as it contributes to a slowdown in access times.

40/ For definitions of CAD, CAM, CIM, etc. see United Nations Economic Commission for Europe (1987), Annex I, pp. 149-158.

41/ See "Inside Technology", Electronics, 29 October 1987.

42/ See "Feature: design automation", Electronics Weekly, 6 July 1988.

43/ See Jackson (1986), p.2.

44/ Automated translation has been a particular focus of interest at the Commission of the European Communities, with the Eurotra project. See Computer Weekly, 9 January 1986. For a survey, see Laub (1986).

45/ For a detailed account of pattern recognition techniques and applications, see UNIDO (1986).

46/ See, for instance, Jackson (1986), and UNIDO (1986), and Gahan (1988).

47/ See "AI woos MIS anew", Computerworld: Focus, 1 June 1988.

48/ For instance, a recent article (Capello et al., 1988) setting out the conditions for a successful expert system project specifically mentions the need for training in the LISP language. For a comparison of LISP and PROLOG see Tichy (1987).

49/ On problems of assessing expert system development see Socha (1988) reported in Data Processing Digest, no.6, 1988.

50/ See Sommerville (1985), p.3.

51/ The best-known work on this subject is Boehm (1981).

52/ The initial successs of Apple as a manufacturer has in fact been attributed to the success of Visicalc. "Survey: Personal Computers", Financial Times, 14 September 1988.

53/ For example, in an advertisement for the DBMS Oracle, the speed of selection, projection, and calculation is compared against that of selection in Informix, another DBMS. (UNIX/WORLD, December 1987). In an advertisement for Turbo Basic 1.1, both the speed of compilation and the speed of execution of the object programme are compared to those of QuickBasic (Microsoft). (Personal Computer World, September 1988).

54/ For instance, the very successful database software for personal computers, dBaseIII, has engendered many other products: see "dBase add-one", Personal Computer World, September 1988. The operating system itself, MS-DOS, has done the same, for instance with "shells" to improve the user interface: see "Eight DOS Shells that Make Computing Easier", Infoworld, Vol.9, January 19, 1987, reported in Data Processing Digest, 3/87. In the mainframe side, a large number of performance monitors are available; one source gives 43 for IBM systems (Xephon Technology Transfer Ltd., Xephon Publications, Autumn 1987).

55/ It is not an absolute control, because to raise rents too much would open the door to a rival producer. But the inconvenience of transferring to another software system would in any case deter many from changing.

56/ Filling gaps, as a strategy, involves an appreciation of many aspects of hardware and software. It may also involve working with hardware directly to such an extent that the operation may seem more like hardware manufacturing. But in fact there is no physical transformation of the materials, apart from the linking of chips and connectors together. The exact pattern of linkage may be unique but it uses printed circuit board (PCB) techniques that are well understood. The real contribution and the source of value added comes from the software, which can be embodied in ROMs, EPROMs, EEPROMs or ASICs.

57/ Representation at these fairs of the new software company is another matter and often an expensive one. The average cost of the space for a stand at CeBIT is DM 200 per square meter, and to this must be added the construction of the stand an many other associated costs. The total expenditures of Siemens in 1989 was 8 million DM. See "Wieviel ausgegeben wird, um Geld zu verdienen", Computerwelt, Nr. 5, 11 March 1988.

58/ See, for instance, Davis (1985). On the terms and conditions of software sales, see UNIDO (1985).

59/ See Boehm (1987) and Sommerville (1985). See also Roman (1986), reported in Data Processing Digest, 11/86, for some examples of COBOL tools for an IBM environment.

60/ "IPSEs start to come into their own", Computer Weekly, 4 February 1988.

61/ See Rubin (1987), reported in Data Processing Digest, 8/87, and also Boehm (1981).

62/ It has been suggested, however, that the maxim "any publicity is good publicity" applies here, but this seems to be more in connexion with large companies who can supply updates at relatively little cost to registered users and thus get a good reputation for quality control. See "What's bugging you", Practical Computing, July 1986.

63/ See "Software Makers Battle the Bugs", Fortune, 17 February 1986.

REFERENCES

Arossa, Lydia. Internationalization of software and computer services. ICCP/OECD Report to be published in 1988.

Bernhard, Robert. Computer architecture: computing at the speed limit. In Torrero, Edward A. (ed.). Next-generation computers. IEEE Press, 1985.

Boehm, B.W. (1981). Software engineering economics. Prentice-Hall, 1981.

Boehm, Barry B. Improving software productivity. IEEE Computer, September 1987.

Codd, E.F. (1970). A relational model of data for large shared data banks. Communications of the ACM, Volume 13, No. 6, June 1970.

Conklin, Jeff. Hypertext: an introduction and survey. IEEE Computer, September 1987.

Cupello, J.M. and Mishelevich, D.J. Managing prototype knowledge/expert system projects. Communications of the ACM, Vol. 31, May 1988.

Dahl, O.-J., Dijkstra, E.W., and Hoare, C.A.R. Structured programming. Academic Press. 1972.

Davis, G.G. Software protection: practical and legal steps to protect and market computer programs. Van Nostrand Reinhold, New York, N.Y., 1985.

Gahan, E. "The impact of expert systems". Industry and Development, No. 23, 1988.

Gimaric, Charles E. and Milutinovic, Vjelko M. A survey of RISC processors and computers of the mid-1980s. IEEE Computer, September 1987, pp. 59-69.

Isaak, J. Designing a standard. Computer World: Focus, Vol. 20, 20 August 1986.

Jackson, P. Introduction to expert systems. Addison-Wesley, 1986.

Kernighan, B.W. and Ritchie, D.M. The C programming language. Prentice-Hall, 1978.

Labich, Kenneth. The shootout in supercomputers. Fortune, Vol. 117, No. 5 29 February 1988, pp. 37-40.

Lamb, J. Translation systems: bridging the language gap. Datamation, Vol. 32, No. 1, 1 January 1986.

Malloy, R. First impressions: IBM's OS/2 extended edition. Byte, July 1988.

Martin, J. An end-user's guide to data base. Prentice-Hall, Inc., New Jersey, 1981.

Mills, H. Structured programming: retrospect and prospects. IEE Software, Vol. 3, No. 6, November 1986.

Nelson, Theodor H. Managing immense storage. Byte, January 1988.

OECD 1985 a. Software: an emerging industry. Information Computer Communications Policy No. 9. Paris, 1985.

OECD, Working Group on Accounting Standards. Working Documents, No. 1: Accounting treatment of software. 1986.

OECD. Standards in information and communications technology. Paris, 1987.

Pyle, I.C. The Ada programming language. Prentice-Hall International, 1981.

Roman, David. Classifying maintenance tools. Computer Decisions, Vol. 18, 30 June 1986.

Rubin, Howard A. A comparison of software cost estimation tools. System Development, Vol. 7, May 1987,

Schware, Robert. Software industry development in the Third World: Policy guidelines, institutional options, and constraints. World Development, Vol. 15, No. 10/11, 1987, pp 1249-1267.

Socha, Wayne J. Problems in auditing expert system development. EDPACS, Vol. 9, March 1988.

Sommerville, I. Software engineering. 2nd Edition. Addison-Wesley, Publishing Country, 1985.

Tichy, Walter F. What can software engineers learn from artificial intelligence? IEEE Computer, November 1987.

United Nations. International standard industrial classification of all economic activities. Statistical Papers Series M., No. 4, Rev. 2. Sales No.: E.68.XVII.8. New York, 1968.

United Nations. Standard international trade classification revision 2. Statistical Papers Series M., No. 34/Rev. 2. Sales No.: E.75.XVII.6. New York, 1975.

United Nations Economic Commission for Europe. Software for industrial automation. UN Sales Publication No. E.87.II.E.19. 1987.

United States Department of Commerce. 1986 US industrial outlook. January 1986.

UNIDO (1986). Applications of pattern recognition and image processing to industrial problems in developing countries. Prepared by TATA Research Development and Design Centre. UNIDO/IS.609, 14 February 1986.

UNIDO (1987). Expert systems: prospects for developing countries. Prepared by A.K. Jain. UNIDO/IPCT.41(SPEC), 23 September 1987.

UNIDO (1988). Software production: organization and modalities. Prepared by Hans-Jochen Schneider. UNIDO/IPCT.63, 17 May 1988.

Weber, J. A window into the brain. Personal Computer World, March 1988.