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.

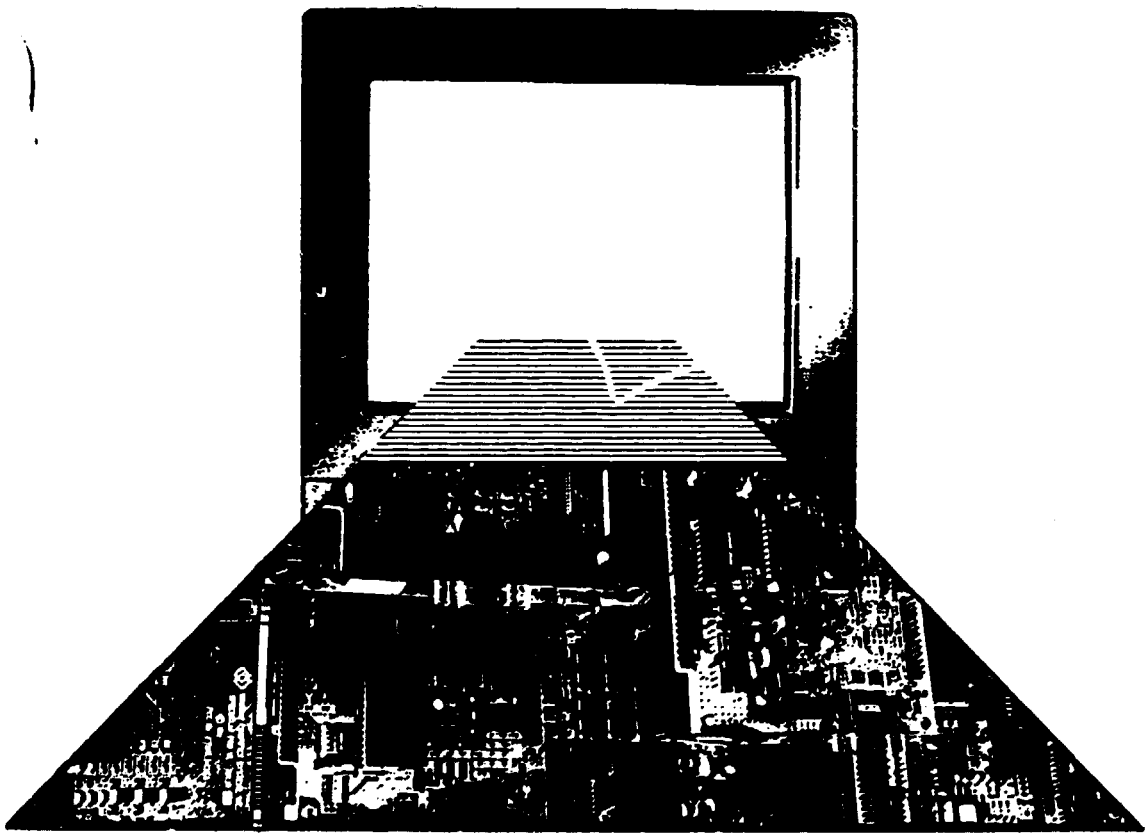
For more information about UNIDO, please visit us at www.unido.org

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

20541

Software Industry

CURRENT TRENDS AND IMPLICATIONS FOR DEVELOPING COUNTRIES



UNIDO

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION

General Studies Series

SOFTWARE INDUSTRY

CURRENT TRENDS AND IMPLICATIONS FOR DEVELOPING COUNTRIES

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
Vienna, 1993

* * *

This study was compiled, and in part written, by R. Narasimhan of CMC Limited, Bangalore, India, under the general direction of K. Fialkowski, Chief, Informatics Unit, and under the supervision of K. Venkataraman, Director, Technology Development and Promotion Division, UNIDO. The author gratefully acknowledges assistance provided by the National Centre for Software Technology, Bombay, and particularly P. Sadanandan, Associate Director, for the preparation of computer readable versions of the UNIDO documents included here.

ID/SER.O/14

UNIDO PUBLICATION
UNIDO.93.2.E
ISBN 92-1-106283-7

PREFACE

In the early 1980s UNIDO recognized software production as an industry and began to promote actions that developing countries could take to develop that industry. The concept has been elaborated through studies dealing with software development and applications for developing countries, their approach to software production and guidelines for organizing software houses. A selection from those studies, written mostly by UNIDO consultants, constitutes the content of this book.

The software that can be developed and deployed determines the speed with which computing systems penetrate and transform industrial and service enterprises to increase the quality and quantity of their outputs. The software and services markets of developing countries are small in comparison to those of the United States of America, Japan and western Europe. These markets, however, are growing rapidly, particularly in Brazil, China (Taiwan Province), India and Singapore.

Today, software production is an industry essential for the growth of the economies of developing countries: launching programmes to promote strong endogenous software industries is a priority task.

Applications of microprocessors that radically enhance the capabilities and functions of products depend on having software available. In many applications the cost of microprocessors is a negligible component of the product price. Functionally, a microprocessor-controlled product is more versatile, has better performance and, in many cases, is more energy-effective. Also significant, especially in the context of developing countries, is that a complicated microprocessor-controlled product needs fewer qualified operators at the technician level. Software, however, is a necessary component of microprocessor systems as it practically enables the system to perform application functions. Therefore, both microprocessor applications and software are also priority developmental tasks for developing countries.

Comparing software production with other modern technology products that could be manufactured in developing countries, the relatively low investment needed for software production is a substantial advantage. The technological equipment required for the production of software consists mainly of a computer system or systems, which in many cases may be rented from the supplier, with the cost of rental even being offset by the income derived from the sales of the software. More advanced software tools, however, are more expensive, at least at present.

For the foreseeable future, labour will remain a priority input to the software development process. Therefore, developing countries may have a comparative advantage in this production based on their lower costs of highly qualified manpower. This advantage will decrease in time with increased automation of software production.

One substantial obstacle for software production in developing countries is the market for the products. The reasons for this are as follows:

(a) The local market is usually limited by the narrow scope of computer applications in developing countries, and it cannot create a broad demand for full-scale production;

(b) The export market, mostly in industrialized countries, is difficult to enter. It requires proper marketing, a product dissemination network and a strategy to overcome unfounded user prejudice concerning the quality of software originating in developing countries.

Observation of the computer market shows the possibility of introducing applications systems in developing countries in an inexpensive way. These systems can support, in a systematic manner, industry, commerce, public administration and banking and insurance companies in the management of masses of data (text processing, database systems, information systems, statistical computation), the control of processes (manufacturing, quality control, optimization procedures) and governmental and industry planning (model building, simulation, optimization).

To ensure appropriate software supply in developing countries, several objectives should be met:

(a) To establish self-supporting software production in developing countries, serving primarily the local and/or regional market;

(b) To establish the training of local staff in advanced programming technology;

(c) To create a basis for developing the local utilization of computers to solve optimization problems, *inter alia*, those of small and medium-sized industries as well as other applications;

(d) To establish mechanisms for the export of software from developing countries.

In developing countries, special informatics applications requirements involving hardware and software could be met by importing the hardware (or components of it, if local systems integration etc. is appropriate) and developing the software locally. These special requirements derive from the following: high unemployment and resulting poverty levels; the small scale of industry allied to the fact that most people live in villages; the lack of adequate infrastructure to support large industrialized centres; and the need to concentrate on rural development as a prerequisite for industrial development so as to generate adequate demand. The potential of foreign-produced hardware (when augmented with suitable locally produced software) for innovative local applications can be best judged by persons

fully conversant with local conditions. Governments can take a number of measures to promote an endogenous software capacity, *e.g.* providing finance for the acquisition of computers, training, etc.; granting tax exemptions on reserves set aside by software houses for use in connection with the inevitable modifications required on software as a result of experience in the field or of material changes in circumstances; and making legislative provision for the copyright of software.

Suitable institutional arrangements to manage and coordinate endogenous software production (including supporting personnel training arrangements) to ensure that the software produced is of a high quality and that it is fully compatible with national application priorities are a prerequisite.

UNIDO efforts are aimed at the following:

(a) Increasing awareness concerning endogenous software production as well as assisting the developing countries in organizing software production. This element was underlined during the discussions at both meetings of the Consultative Group on Informatics Technology for Development held at Vienna, in March 1984 and December 1987. Several UNIDO documents ("Problems of software development in developing countries" (UNIDO/IS.383), "Guidelines for software production in developing countries" (UNIDO/IS.440), "Software engineering: a survey" (UNIDO/IS.446) and "Software production: organization and modalities" (UNIDO/IPCT.63)) were prepared. For countries less advanced in the field of software, a special approach was worked out (" The prospects for software production in developing countries" (ID/WG.437/4)) and adopted at the National Meeting on Applications of Microelectronics and Software, held at Nairobi, 18-23 February 1985. For applications of informatics in small and medium-scale industries, "Guidelines for the diffusion of informatics in small and medium companies (SMC)" (IPCT.81) was prepared, and the Meeting of the Consultative Group on Informatics Technology for Development, held 11-13 December 1989 at Buenos Aires, dealt with the subject, stressing the importance of regional co-operation in informatics (Report of the Meeting of the Consultative Group on Informatics Technology for Development (IPCT.109) and "Situación de la informática en América Latina puntos de partida para una acción regional" (IPCT.111));

(b) Assisting in the creation of national policies on software development "Guidelines for software development in developing countries" (UNIDO/IS.439) was prepared and disseminated among decision-makers in the developing countries;

(c) Devoting attention to the acquisition and protection of software. "Trends in commercialization of software in developing countries" (UNIDO/IS.574), "Recent trends in contractual practice relating to acquisition of software in the United States of America and the EEC" (IPCT.139) and the "UNIDO/REMLAC preparatory report on the elaboration of a programme of action aimed at strengthening the negotiating capabilities in the acquisition of hardware and software" were prepared. Their recommendations influenced governmental

policies especially in the member countries of the Regional Network for Microelectronics for Latin America and the Caribbean (REMLAC). (16 countries from Latin America and the Caribbean are participating in this programme.) "Informática y desarrollo industrial en América Latina" (IPCT.136) was also prepared for REMLAC.

As a result of the experience gathered and the articulated needs of developing countries, UNIDO now concentrates on assisting developing countries in the creation of an infrastructure and in working out modalities for endogenous software production with export possibilities to industrialized countries. The first phase of the regional programme, "Preparatory Assistance for Regional Project for Strengthening Microelectronics Infrastructure and Capabilities in Member Countries for a Regional Network for Microelectronics in the ECLAC Region" (DP/RLA/86/003) has been implemented. The second phase, "Cooperación en informática y microelectrónica" (DP/RLA/92/014), directed mostly to software and applications in industry, is now under implementation.

There is a need for short- and long-term measures on the part of developing countries. Short-term measures include the organization and/or strengthening of associations of software producers. The most important requirement in the long term is that for skills. Four types of skill are required:

(a) The first is the skill required for identifying and specifying applications of software;

(b) The second type of skill, and the predominant one for development, is that for software identification, adaptation and development;

(c) The third type of skill is that for promoting the widespread adoption of computer systems among local users;

(d) The final type is that for the service and maintenance of software.

The development of appropriately qualified human resources constitutes a critical condition for computer applications.

In this context, UNIDO attempts to identify the content and modalities of a possible framework for action in the field of software production and applications. The following could fall within that framework:

(a) A set of government policies for the development of an endogenous capability in this field;

(b) Specific programmes of action, such as in the field of human resource development and other relevant areas, preferably in the form of a plan of action to be implemented by the concerned agencies and institutions;

(c) Possible institutional arrangements for developing endogenous capability, such as associations of software producers or a centre for software applications to industry.

In the scope of international cooperation, UNIDO provides technical assistance and advisory services for the establishment of software production enterprises and associations and for the development of human and institutional capabilities, in particular for applications of software to industry. Applied research and development in the field of software related to microelectronics and its applications may be the subject of cooperation. In the field of public domain software, UNIDO promotes the exchange of information among the countries and provides advisory services as required. UNIDO also provides advisory services for policy formulation, if requested, including the preparation of programmes and plans.

Now, in the early 1990s, software technology is being recognized as a new and important sector deserving development in itself. However, in the context of UNIDO activities, not all the elements of the new sector necessarily deserve attention. The most important elements of informatics technology from the point of view of UNIDO may include the following:

(a) Industrial software applications;

(b) Software as a small-scale industry;

(c) New production technologies for software;

(d) Standardization and legal aspects in both hardware and software;

(e) Monitoring the development of software technology.

In many cases software applications brought to developing countries did not produce the expected results, nor has the anticipated diffusion for development purposes been realized. Therefore the elements of the UNIDO informatics programme attempt to identify more precisely the areas and environments of industrial applications that are most crucial for development. In the context of developing countries, the following applications seem to be of special importance:

(a) Software applications designed to improve the efficiency of industry;

(b) Software applications designed for small and medium-scale industries;

(c) The production of software for export.

It is hoped that this book will contribute by bringing together information on the different aspects of software development.

Contents

V. Implications for Developing Countries

VI. Conclusion

I

Overview

Software Industry: A Developing Country Perspective

R. Narasimhan*

1. Introduction

Software is the lifeblood of informatics. Without appropriate and efficient software, information processing systems are so much lifeless hardware with only scrap-value. Software availability, therefore, assumes strategic importance in the promotion and use of information technology (IT) in developing countries (DCs) for development-catalyzing applications. As Correa (1990) notes: "The establishment of a dynamic software industry has been identified in many DCs as essential to meet the needs of a growing number of domestic users of informatics systems. In some of these countries software has been viewed moreover as opening up new and substantial export opportunities." As part of its Technology Development and Promotion Programme, UNIDO has been promoting the use of software by, and the development of software industry in developing countries. This book brings together a selection of reports by experts for dissemination by UNIDO addressing a variety of issues that arise in trying to promote software production and use in developing countries.

In this introductory chapter**, we shall try to set the stage for the chapters that follow. In the first part of this chapter (sections 2 and 3), we shall discuss briefly the nature of the software industry and the development and growth of this industry in the industrialized countries — mostly in the United States of America. Can the developing countries use the experience of the developed countries as a model while trying to develop the industry in their own countries?

Next, in Sections 4 and 5, our focus will be more centrally on the developing countries. We shall discuss the issues that arise in meeting their internal software needs and also in becoming successful exporters of software products and services. Can globalization (i.e., giving a predominantly export-orientation to the software industry) succeed as a methodology for building a viable software industry in the developing countries? We shall argue that the development of an informed, diversified, and competitive domestic market is a prerequisite to effectively meet the internal needs, and also to succeed in export. In fact, internal use of software to support development-catalyzing applications is the greater need of the developing countries. It would be a fatal mistake to treat the software industry in a developing country as an island interacting with the outside world only in order to earn

* Professor of Computer Science, Computer Maintenance Cooperation Limited, Bangalore, India

** Portions of this chapter have been adapted from Narasimhan (1984) and Narasimhan (1992).

foreign exchange. This industry could, in fact, generate returns more effectively by contributing to the competitive advantage of other industries in the domestic, as well as the global, markets. (Atul Wad, in his paper included in this volume, develops this thesis in greater detail.)

In Section 6, we shall discuss some development-catalyzing applications which are intrinsically important to the developing countries and which, at the same time, offer opportunities for innovative use of IT. We shall see that efforts along these lines are essential if a sizeable domestic market for informatics industry in general, and software industry more particularly, is to be developed.

In Section 7, in conclusion, we shall list some initiatives that would assist developing countries in building up a viable software production base and a market for software goods and services.

2. Software: An Engineering Industry

A typical engineering industry is concerned with production in large quantities of a specific item. This is true whether the concerned end item is a discrete hardware (automobile, computer) or a bulk material (chemicals). Typically, a factory delimits a production environment within which this quantity production takes place. To realize copies of the identified end-product (or bulk quantities of it in the case of chemicals, etc.), various input materials are taken through a complex network of physical processes. Specific classes of tools (machine tools, chemical processing equipment, and/or people with hand-tools) are deployed appropriately to carry out the processing at the various stages.

Technology transfer in such a situation can be summed up as consisting of transfer of know-how and information concerning the following facets:

1. Factory planning
2. Setting up production facilities, providing tools, plant machinery and equipment
3. Technical documentation
4. Manufacturing processes
5. Quality assurance and inspection methods
6. Manpower training.

At a later stage one may want to consider the transfer of know-how to create new engineered products of the same type (a new automobile, new computer designs, new chemicals, etc.), and establish appropriate variants of the original production process so that quantity production of this new item can be realized. The transferred know-how could, in the best of cases, then be expected to go from

Local Manufacture $\longrightarrow$ Local Engineering $\longrightarrow$ Local Design

or in other words,

Manufacturing know-how ⟶ Engineering know-how ⟶ Design know-how.

In the case of a software product, in contrast to the conventional engineering products as discussed above, once a software product has been designed, coded, tested and found to work, making copies of it is a trivial process if the copy is to run on an identical hardware. A non-trivial conversion process has to be gone through if the same software routine has to be made to run on a different hardware. Since hardware keeps changing all the time, and customers like to opt for improved hardware as these become available (through technological innovation), already running software products need to be converted to run on new hardware. The conversion market is thus an ever present and conceivably lucrative market for a software production agency.

But the legitimate objective of software production cannot be either copying or converting, but creating *ab initio* the first version of a software package. Since the software package is produced only once, a production facility set up for software production has more the character of a design centre rather than that of a conventional factory. This design-intensive character of software technology makes a software production facility resemble more closely a microelectronics fabrication facility rather than traditional engineering factories. This is because in both these cases, a production facility, once established, is much less tied to the production of a particular engineered item than is the case with conventional engineering factories. Hence, to be able to exploit the full potentials of a production facility in these two cases, design know-how to design new products is an essential requirement. From this it follows that technology transfer in the case of software technology has to be necessarily at the level of transfer of design know-how. This is intrinsically a difficult task and is, besides, predicated on the availability of appropriately qualified professionals who are capable of absorbing the design know-how when this is transferred.

The early efforts at software generation were really small-scale R&D efforts. Software packages were created using tricks, clever stratagems, and *ad hoc* techniques, by clever individuals or small teams of highly knowledgeable persons. This methodology was acceptable so long as the software created was used by the designers themselves or by an equally well-informed in-group. But as the complexity of the software products increased, larger teams of persons had to be involved in creating them. Also, these products had to be increasingly created for wide distribution, for use by a variety of persons with little or no knowledge of the design process involved. Under these circumstances, the earlier informal, *ad hoc*, production techniques could no longer work. These had to be replaced by more formal and systematized design, coding and testing processes. Precise and detailed documentation assumed great importance both at the design and coding stage and, more importantly, for the continued maintenance of the software products. Also, when a production team was concerned with designing and producing a wide variety of software products,

the problem of improving the productivity of the personnel began to assume increasing importance. The incorporation of production tools (their design and their appropriate development) became a necessity [Martin (1986), Chorafas(1986), Varsegi(1990)].

The software production process thus increasingly began to assume the character of an organized engineering industry. The software industry as it currently flourishes in the advanced countries, thus has many of the ramifications of more conventional established industries. Nevertheless, its basic character remains quite distinct from that of a factory-based manufacturing activity. This is essentially due to the intrinsic design-level orientation of the activity.

The software production process construed as an engineering activity is described in greater detail by Turski in his paper included in Part II of this volume. In a companion paper, Schware discusses at some length the organizational and management aspects of software production.

Since software technology and engineering are still going through rapid developments even in the advanced countries, it should be useful to acquire some appreciation of the historical trends in this development. This would enable one to understand the kinds of pressures that have given shape to the software activities as we find them in the developed countries today. The products that are being created and used in these countries, and the structures that have evolved to create these products, may not all be immediately relevant to developing countries. We must have a clear perception of the essentials and the accidentals. Future trends in the developed countries are of some consequence to developing countries in the context of export and trade. Hence, it is of importance to understand how things came to be the way they are in the software sector in developed countries and where things are headed in the immediate and foreseeable future.

3. Software Industry: Developed Country Experience

3.1 Historical Background

Computers in the developed countries had two distinct origins — one concerned with sorting and tabulation of census data, and the other concerned with the computation of numerical tables. Hollerith in the United States and Babbage in the United Kingdom of Great Britain and Northern Ireland were the founding fathers of these two motivated efforts. The relevance and importance of Hollerith's ideas to commercial data processing applications were perceived fairly early on, and organizations like the National Cash Register Co., and, somewhat later, International Business Machines (IBM) were interested and involved in supporting and further developing these ideas into commercially viable products. Babbage's ideas and early attempts, on the other hand, were far ahead of his time and eventually had to be abandoned without realizing a working model. But this motivation to build

mechanical devices for computing numerical tables acquired fresh impetus in the early days of the Second World War and rapidly resulted first, in electromechanical, and later electronic, computers.

These two streams of efforts continued to develop independently in the period 1940-1955; one punchcard-based and mostly electromechanical, concerned with business applications and market and sales-oriented; the other papertape-based, using vacuum tubes, concerned with scientific computations and academic- and research-establishment-oriented. But in both these streams, there was not much difference between hardware efforts and software efforts — the same people were more or less involved in both these areas In many cases, end-users formed part of these teams and helped in writing the application programs.

Today, the whole picture has changed completely. Hardware manufacturers, software manufacturers and end-users have become three distinct groups. The end-user does not depend on the hardware manufacturer to supply all the software, application packages and services that he is in need of. The software manufacturer, analogously, does not produce software necessarily for one hardware manufacturer exclusively or for one class of end-users. This state of affairs assists the hardware manufacturer also, since it is not essential now for him to invest in software and application development in order to be able to sell his hardware to the end-users. No doubt, this is somewhat of an idealized picture: this is mostly true in the case of microcomputers (PCs and workstations), and to a large extent true in the case of minicomputers. This is perhaps less true in the case of mainframes, where a large part of the software effort — certainly for system software, and in many cases also application software — still belongs to the hardware manufacturer. However, it is in the context of this devolution of market responsibilities that software development has acquired the characteristics of a self-contained industry.

It is important to realize that the software industrial profile that we currently see in western Europe and the United States has not come about based on any planned growth or systematic design. Various kinds of market pressures and demand pulls have provided impetus to the growth of the industry and the particular kind of structuring one finds there. In the very early stages, the hardware manufacturer provided all the system software and much of the application software. All this software was not specifically costed, but sought to be absorbed more or less in the total system price. Soon it became evident that software development and maintenance costs had become equal to, and were beginning to exceed, the hardware cost. It thus became essential to assign a commercial price to software and "unbundle" the sale of software and hardware. IBM's decision to adopt this strategy in 1969 contributed significantly to the creation of a separate software sector since, in principle, unbundled software did not have to be procured from the hardware supplier himself.

In the 1960s, defence and space software needs in the United States opened up vast opportunities for independent software groups to undertake to develop the specialized software needed by these end-users on the basis of development contracts (mostly on a cost-plus arrangement). Much of this software was extremely large, complex, and had to

meet stringent reliability and real-time constraints. Opportunities to work on these development contracts were of extreme value to the independent software contractors in evolving software production methodologies to manage large, complex software projects, and ensure quality and reliability of these products. These tested methodologies were then spun off to the commercial market to form standard industry practices

Extremely rapid developments in hardware technology due to progress in microelectronics technology resulted in keen competition among hardware manufacturers to come out with newer and newer generations of hardware — especially in the minicomputer and microcomputer categories. This widened the gap between hardware and software availability in the market. Not being able to cope with the problem of bridging this gap, these hardware manufacturers had no alternative except to accept and even encourage the emergence of independent software entrepreneurs. For these entrepreneurs, the ever widening information services market proved to be an ideal entry-level business because of its relatively low entry-cost, high rates of return and profit margins, and the great diversity of software requirements and the consequent impracticability of any one firm, or even a few firms, dominating the market. The software industry, as it exists now in the developed countries, is thus a highly fragmented one spanning a very wide spectrum of products and services, and a very wide range of sizes of units and incomes. Some idea of the diversity of products and services that make up the software industry may be obtained from Table 1. We shall consider presently, in more quantitative terms, the structure of the software industry. The software market is discussed in greater detail by Kopetz in his paper that constitutes Part III of this volume.

3.2 The Structure of the Software Industry

The Organisation for Economic Co-operation and Development (OECD) in *The Internationalization of Software and Computer Services* (1989, p.16) defines the market segments of the software and computer services industry as follows:

a) *Software* that can be supplied in three different forms:
   1. As *packaged* mainly considered as an intangible product,
   2. As *customized* or created *ad hoc* for each client, considered as a service,
   3. As *turnkey systems software*, or offered as part of a turnkey system (hardware plus software) designed for specific clients. Providing turnkey systems is considered as a service;

b) *Professional Services* that include consultancy and training, normally this category includes custom software development as well;

c) *Processing Services* that include all services related to data treatment, both on-line and stand-alone. In the former case, VANS (value-added network services) and data-base enquiry are included. Facilities management is included in this category.

Obtaining reliable data on the magnitude of the software and services market — either at the level of individual countries, or globally — has proved to be extremely difficult. This is because, apart from the software and services that are traded in the open market, sizeable software development activities take place in-house, and additionally, contract software development is carried out and "traded" within corporations and multinational companies. These are, for the most part, unaccounted for. Market research firms that normally collect and publish data on the software market use methodologies that are often haphazard and unsystematic. Schware (1989) cautions that "market research reports, particularly in the software and services industry, need to be treated with a healthy degree of skepticism."

Table 1. World Market Revue

	1988 %	1993 (Projected %)
Processing Services	30	21
Professional Services	34	34
Software Packages	30	36
System Integration	6	9
Total (value)	US$ 110 billion	US$ 225 billion

Source: Kohli (1991)

In the absence of comprehensive and completely reliable data, rough estimates arrived at through various *ad hoc* methods are widely quoted and made use of for purposes of discussion. Keeping in mind the caveats made earlier, one such estimate of the market is given below:

Concerning the global distribution of the software and services market, OECD (1989, p.10) notes: "Few estimates exist of foreign penetration in country markets and almost all tend to provide evidence of the strong position of United States firms (especially in European countries and Japan) as well as the high level of import dependence. .... At present, trade mainly occurs inside the OECD area, but interest in other country markets is growing, as demonstrated by the increasing number of joint ventures for software development, especially by Japanese firms." The OTA (Office of Technology Assessment) estimates — as quoted in Watanabe (1989) — the world market share of United States software suppliers, including the foreign affiliates of United States firms, at about 70 per cent in 1985. The global revenues of United States suppliers of packaged and custom software is believed to have grown to US$31 billion by 1987. Foreign sales accounted for 40 per cent, having increased faster than domestic sales. The contribution to the world trade of software firms from newly industrializing economies (NIEs) is negligible and their earnings are marginal.

Today, the software industry, particularly in the United States, is in a state of flux. According to the OECD (1989, p 9): "The structure of the software and services industry is dominated by buyouts and selloffs. .... Information on acquisitions is available for the United States market only, but there is evidence ... that this activity has also been important in European markets; and it is occurring internationally." Large firms like IBM — which once considered itself strong enough to "go it alone" — are recognizing the need to cooperate with independent software vendors.

The software industry has traditionally been seen as fertile ground for innovation and small-scale entrepreneurship. Fairly low barriers to entry in terms of capital requirements facilitate business start-ups. However, as OECD (1989, p.9) cautions: "It is not easy to develop an activity to a meaningful size. A particularly sensitive area is that of packages, which are products that require marketing ability and the availability of funds. Thus, despite ease of entry, barriers in terms of managerial ability and funds availability may hamper growth. For small niche companies and start-ups, the increasingly logical evolution seems to be acquired. Large software developers grow by extending their line of offerings and absorbing pioneering breakthroughs." Commenting on the increasingly uphill efforts faced by start-ups, *the Economist* (21 September 1991) observes: "Even successful niche companies will find it impossible to grow as big as giants like Lotus Development, Computer Associates, or Microsoft without breaking out of their niche. Doing that is certainly harder now than it was a decade ago. But that does not mean niche firms are not stable, profitable businesses."

Welke (1980) in an early survey of the "Origins of Software" identified the following as factors contributing to the phenomenal success of the software market in the United States. These factors remain as important today as a decade ago.

1. Leasing instead of selling: software is leased or licenced and not sold.
2. Pricing maintenance separately: continued maintenance of a package is separately priced from the basic one-time sale price — it could be anywhere from 10-15 per cent of the basic sale price.
3. Long-term vendor-customer relationship: an end-user expects a long-term commitment from the software vendor to keep the package updated and contemporary.
4. Designed to minimize installation costs: software package design must have built-in features to minimize installation costs (which normally involve some special tailoring to match the end-user's special constraints and needs).
5. Avoiding over-designing and over-selling: reliable operation and continued support are more important for a successful market than producing an over-designed product or mounting a hard-sell campaign.

The development and growth of the software industry simultaneously prompted the development of protection measures to guard against unlawful trade practices involving software products. In the early days, three basic protection methods were resorted to: trade secrecy, patent law, and copyright law. However, most countries have now come to accept the copyright law as the most appropriate and enforceable method of protection. Copyright law notwithstanding, software piracy — especially of package software intended for use with microcomputers (PCs) — continues to pose a major threat to the healthy growth of the software market. An article in *Financial Times* (10 December 1991) estimates worldwide annual losses suffered by software companies due to piracy to be US$12 billion — half of this attributable to software piracy in Europe. Illegally copied software in the OECD countries is estimated to range from 45 to 90 per cent. Legal issues relating to software development, procurement, and distribution are discussed in great detail by Soltysinski and Correa in their papers, which make up Part IV of this volume.

It is predicted that the software market is bound to maintain its phenomenal growth for the following reasons [Welke, 1980]:

1. The market is nowhere near saturation. New applications are all the time becoming economical as the hardware and communication technologies advance and prices fall. The volume of computing is continuing to grow.
2. The business office environment is becoming information-processing oriented with the increased use of intelligent office equipment and tele-informatics.
3. The use of databases, both centralized and distributed, is increasing in business and industry.
4. Analogous to office automation, factory automation is also advancing at a fast pace. Robotics and the use of intelligent tools are likely to become norms in the factories of the future.
5. Integrated digital communication services (voice + picture + text + data) would widen the base for software products and services.

While one does not see any limits to the demand for software products and services in the foreseeable future, the shortage of manpower supply to produce all this software is already a live issue in most of the developed countries. Watanabe (1989), quoting an OTA study of 1987 notes that 600 to 700 thousand programmers in the United States in 1983-1984 were already unable to cope with the software demands of that country. Their salaries were steadily rising at an annual rate of 10 per cent from 1978. A Ministry of International Trade and Industry (MITI) sponsored study published in 1987 anticipates a 'software crisis' in Japan as a result of the estimated growth of demand for software personnel from 430,000 to 2.15 million between 1985 and 2000. According to a Department of Trade and Industry study (1987), of about 45,000 persons employed in the software and computer services

industry in the United Kingdom, only about 1,000 are project managers with experience of large software projects; half of them are managing companies rather than managing projects.

This increasing shortage of qualified software professionals in the developed countries was seen as providing opportunities for developing countries to make an entry into the global software market. It was believed that their high quality surplus manpower resources could provide a competitive advantage in the market provided adequate training and transfer of know-how were arranged and appropriate trading channels were created. Experience during the past decade has, however, shown that people-availability in itself does not help very much. As discussed earlier, software development is increasingly becoming an engineering activity. To compete effectively in the global market appropriate technology support is needed. Tooled environments have to be created and the software professionals have to be trained to function efficiently in these environments. These are capital-intensive undertakings not within the means of many of the developing countries. We shall return to this topic and discuss it at greater length in Section 5 below.

4. The Software Needs of Developing Countries

In the previous two sections, we have considered in some detail the nature of the software industry and the current status of this industry in the West stemming from its historical origins. What are the implications of this picture to developing countries? Software technology is of great importance to the developing countries to meet their internal development-catalyzing application needs. How can they acquire the requisite know-how to deploy this technology effectively to assist development? What can they learn from the experience of developed countries in structuring the software industry in their own countries? What problems would they have to contend with if they wish to successfully penetrate the global software market? These are some of the issues we shall address in this and the next section.

Broadly the internal development-catalyzing needs of developing countries could be grouped into four categories:

1. Applications that relate to socio-economic programmes concerned with basic needs (primary health, literacy, rural employment, housing, transport, water, etc.) and the agriculture sector;
2. Applications in the government sector;
3. Applications that assist in increasing productivity in the industrial sector (goods as well as services); and
4. Applications relating to export and the international software market.

We shall discuss in this section the nature of the applications in the first three categories. Our emphasis will be in analyzing to what extent these applications are unique to the developing countries, and hence require local initiatives to generate the requisite software. In Section 5, we discuss the problems that developing countries have to come to grips with before they can hope to compete effectively in the international market.

Focussing now on the internal application needs (i.e., applications in the first three categories), we shall see that they require software support of two different kinds: (1) software products (packages); (2) software-supported systems. In the first case, if the needed products are available, one could think in terms of obtaining them for immediate use — through purchase or lease. Modifications would still be needed to existing products before they become practically usable (*Datamation* 1980). But in general, this is the simpler of the two cases to cope with. Software-supported systems, on the other hand, almost always have to be tailor-made to suit the specific end-use environments. Even if similar systems are in use elsewhere, transporting them, modifying them, and fitting them to match local needs, may not be easy. In fact, in many cases, creating systems *ab initio* taking into account, in the design stage, local specificities may be a preferred solution. The structures and expertise needed to create such systems may have to be dealt with on a case by case basis. Also transfer of know-how in these system design areas is likely to be less straightforward. But precisely these application areas are the ones of great immediate importance to developing countries. It is in meeting the needs in these application areas that available software production models in the developed countries are likely to be of little relevance.

Existing models in the developed countries are likely to be more valuable and applicable in the third category. But, even in these cases, local conditions and contexts may require the creation and use of new structures. Industrial production establishments in the DCs — even in the more advanced ones — seldom have the level of informatics awareness that is usually taken for granted in such establishments in the developed countries. Information processing practices, and software packages created to implement such practices in the developed countries cannot, therefore, be transported to developing countries and made to function effectively in a straightforward way. Software practices, such as distributed processing and word processing, assume new dimensions in countries where the telecommunication infrastructure is undeveloped, or where the local script and mode of writing differ radically from those in European countries.

Creating structures and transferring methodologies for software production are predicated on the availability of manpower with the requisite training and background in software engineering, knowledge of application areas, expertise in consultancy and systems analysis and, not the least, marketing skills. The methodologies for manpower creation, again, cannot be directly transferred from the developed to the developing countries. Development in the developed countries has been the result of a historical, evolutionary process. This applies to skill and knowledge development also. The developing countries are trying *to achieve development in a broad front simultaneously*. The needed manpower also has to be

developed at many levels simultaneously. A variety of approaches would have to be resorted to at the same time. It is therefore important to determine what are the varieties of manpower development mechanisms needed to be created to achieve this skill and knowledge development at many levels simultaneously and over a short period. It should then be possible to identify the kinds of inputs from developed to developing countries that are likely to have maximum *practical* value in the manpower development area.

4.2 Basic Needs and Agriculture

The priority concern of a developing country is meeting the basic needs of that section of its population, which for various reasons is poor, underprivileged and economically unable to compete under open-market conditions to better its lot. In most developing countries this section of the population tends to live in rural areas, in small, mostly economically unviable and geographically isolated communities. Tradition-bound, lacking in formal education and marketable skills, this population tends to eke out a subsistence by living off marginally productive lands.

Developing countries committed to developing the human potentials of this section of the population to the fullest extent possible and enabling them to contribute effectively to the economies of their countries, have an extremely complex problem to tackle. These countries have to plan and implement a coordinated programme of field-level activities that simultaneously address the following issues as they relate to this depressed section of the population:

- provision of primary health-care and essential education
- provision of housing and water for drinking and cultivation
- improving the economic skills and nutrition
- provision of capital infrastructure to launch individuals and families on economically remunerative activities
- integrating the communities with the institutionalized financial and market structures of the country

and so on. While addressing these immediate and local issues, integrated rural development plans must be oriented towards long-term improvement of the non-urban areas in order to bring them into the mainstream of economic activities and make them productive and self-sustaining.

The implementation of integrated rural development programmes of this kind assume great complexity because of the following factors:

1. the large size of the population to be serviced
2. the multiplicity of agencies involved in providing the services
3. the close coordination between the agencies that has to be maintained and operated

4. the long time periods over which these integrated activities have to be maintained and operated

5. the necessity for ensuring proper temporal sequencing of the varieties of subactivities involved in the projects

6. the great geographical distances that have to be covered to bring a sizeable population into the planned action .

It is precisely in coping with these complexities that informatics technology could play an extremely effective role. As an instrument of management at the rural level, it could provide an integrated framework of assistance to the government to plan, formulate, administer, monitor and control the varieties of programmes that make up the integrated rural development projects. Management information systems that can play this role have to be tailor-made to meet the requirements of each country, and possibl, each subregion in a country. By integrating computing and communication requirements of management in one framework, such management information systems for the rural sector could radically transform the style of government functioning in developing countries. Clearly, this application area is specific to developing countries and models cannot be imported from developed countries since analogous problem-areas do not in general exist there. Expertise in system analysis and software development for coping with this application area must necessarily be indigenously grown within the developing countries themselves. There is much scope here for cooperation between developing countries in the design and engineering of systems.

In the more developed agricultural sector, a variety of applications arise: in land use planning, water management and irrigation control, weather monitoring and forecasting for farmers, and so on. Several implemented accounts of these applications are discussed in the published literature and specialized conference proceedings.

4.3 Government Sector Applications

Every government requires systematic maintenance of records — of property holdings, of transactions of all kinds, of productions and consumptions, exports and imports, and all kinds of indicators of the status and functioning of the economy in general. Foreign reserves, trade balances, educational statistics, consumer spending, price indexes and similar information are essential to monitor and control the performance of the various ministries and departments of the government.

Informatics technology can be readily deployed to service all such applications. Software needs span tabulation, storage, retrieval, of large quantities of data; indexing, classifying, and sorting; and statistical computations of various kinds. More sophisticated applications such as modelling, planning and simulation would call for more complex software production and computer usage. Graphics capabilities of various sorts would be needed in surveying, map making, land-use planning and analysis, etc. Budget planning and control is another vital area of importance.

4.4. Industrial and Service Sector Applications

Informatics technology can be deployed to increase productivity in practically all areas of industrial and service activity. Applications to serve this purpose are more or less well-developed and in active use in technologically advanced countries. Where the technology and organizational set up supporting these activities in a developing country are identical to those in developed countries, application packages should be directly transportable from the latter to the former. Where the technology and/or the organizational set up differ, existing packages may have to be modified more or less extensively.

Identifying priority areas for informatics support is the real issue that developing countries have to face and resolve. They should then be able to profit from the experiences of the developed countries in these areas.

4.5 Creating a Software Industry to serve Domestic Needs

The application needs identified in the earlier sections demonstrate that while genuine and, from a developmental viewpoint, critical needs exist, they remain implicit and passive and do not automatically result in the growth of a demand-driven market. Market development, therefore, is the real issue. The problem is to analyze how a viable domestic market can be developed for software applications. The problems that arise here are not purely commercial, economic, or technological, but also cultural in a deep sense. The cultural barriers are more difficult to penetrate, the more traditional the occupational categories are. However, precisely these sectors need to be made more informatics-conscious if a sustainable domestic market is to be developed for information goods and services. This is so because the vast majority of the population in the developing countries function within these sectors. In Section 6 we shall discuss some initiatives that could help in opening up these sectors for market penetration by IT suppliers. But before doing this we shall, in the next section, analyze whether a viable software industry can be developed in the developing countries primarily addressing the international software market.

5. The Export Sector

5.1 Problems of Globalization

We discussed briefly towards the end of Section 3.2 earlier that the increasing shortage of qualified software professionals in the developed countries was seen by the developing countries as an opportunity to make a successful entry into the global software market. The comparatively low cost of professional manpower in the developing countries was seen as a major competitive advantage in this context. During the last decade, many countries and areas in Asia — including China, India, Republic of Korea, Singapore and Taiwan Province of China— have been trying to formulate national level policies to actively promote the growth of a software industry in their respective countries with a predominant export-orientation. Similar efforts have been mounted by several of the Latin American countries such as

Argentina, Brazil and Chile (see Correa, 1990, for details). However, the experience of these countries over the years has shown that the availability of low-cost professional manpower in itself does not provide a sustained advantage to succeed in the global software market. As Correa (1990) notes: "Prices may be an element to consider when selecting bids. But previous work and the nature of projects the bidding firm has executed may actually be more important in the final selection." Surveys of software producers in the developing countries have repeatedly shown that constraints such as the highly circumscribed local market, the lack of marketing strength, low availability of capital and the consequent restricted potentials for R & D work, and so on, act as major impediments to effective performance in the global market.

Two other reasons for the increasing difficulties faced by the developing countries are the following: (1) increasing competition from the industrialized countries themselves; and (2) the changing technology of software production. We have already considered briefly in Section 2 earlier the changing technology trends in the software production process. We shall discuss in this section some of the implications of these changing trends to successful market penetration by the developing countries.

Mainframe vendors who earlier used to concern themselves only with system software development are increasingly moving into application software development on perceiving the vast market opportunities that are opening up. Application software backlogs are piling up. The market opportunities created by such backlogs are inducing a growing number of United States and west European software firms and non-software firms (such as aircraft manufacturers and financial service providers) to set up off-shore facilities for application software development using the low-cost skills available, or readily trainable, there. What was earlier seen as an advantage to the software firms in the developing countries — namely, the low cost of their professionals — is thus beginning to be exploited by software firms in developed countries taking advantage of their superior skills in management and marketing supported by larger capital resources.

Secondly, as discussed earlier, software development is beginning to be seen as really an engineering effort and, consequently, it is beginning to be appreciated that software industry, to be viable, must be structured and managed as an engineering industry. A major assumption in the management of software development as an engineering effort is that the quality of a piece of software is largely governed by the quality of the process used to develop and maintain it (see the paper by Turski included in this volume).

Automation, based on the use of tools, in the various stages of development of software is an important aid to improve productivity and also the quality of the software produced. Software metrics for quality assurance of the software development process, rapid prototyping through the use of application generators, automated documentation procedures, and other components make up the tooled environment providing competitive advantage in

software business (see the paper by Schware in Part II of this volume). As Schware (1989) points out: "As the software industry becomes less labour-intensive, the quality and availability of skilled labour will become more important than labour costs."

What are then the options open to developing countries aspiring to build up export markets? Schware (1989) suggests five options:

1. Try to position oneself in a knowledge-intensive niche market not easily penetrated competitively by large firms
2. Modify existing application software
3. Acquire and adapt public-domain software
4. Acquire/develop and learn to use tooled environments for software development
5. Form alliances.

The opportunities offered by public-domain software for value-addition by firms in the developing countries are discussed by Botelho and Schware in their papers in Part V of this volume. The paper by Kopetz in the same part discusses some possibilities for the successful development by developing countries of niche markets in one particular area — the production of intelligent products.

The importance of the availability of a diversified, competitive, and informed domestic market is not often realized in building up export capability. A variety of skills — entrepreneurial, managerial and software engineering — are needed to successfully execute software jobs in a highly competitive and technologically fast-moving global environment. The learning involved in acquiring such skills is possible only by servicing an appropriately structured and demanding domestic market. A large pool of highly trained and motivated software professionals can be sustained only by deploying them to work on challenging jobs in the home market. The ready availability of such a manpower resource pool is a prerequisite for effectively competing in the foreign markets. A flourishing domestic market is also a *sine qua non* for building up the capability to innovate, as we shall discuss in the next section.

5.2 Innovation: the Key to Global Competitive Advantage

Michael Porter's (1990) impressive ten-countries study to determine the factors responsible for the competitive advantage of countries is directly relevant to the issues we have been discussing to build up an export market in software by developing countries. The central theme of Porter's study is that the right question to ask is not why some countries succeed while others do not in a global competitive market, but why specific industries located in some countries succeed while the same industries located elsewhere fail. What then are the conditions for a software industry located in a developing country to succeed in the global market according to Porter's analysis?

Firstly, according to Porter's definition an industry is competitive in a country provided it is able to achieve high levels of productivity and increase it over time. This, he claims, is only feasible if the economy allows the industry continuously to upgrade itself. The country's firms must be moving into more and more sophisticated segments of that industry all the time because that is where productivity increases can be maintained.

In the long term, two factors determine the success of a firm's products and services relative to its competitors. They are: lower cost (for given functionalities) and improved functionalities (i.e., improved quality, better features, better after-sales service, and so on). It is this second factor (which Porter calls 'Differentiation') that allows a firm to command premium prices leading to superior profitability.

The most important prerequisite for a country's firms to gain competitive advantage is to function in an aggressive and competitive demand-driven environment. Porter emphasizes: "Among the strongest empirical findings from our research is the association between vigorous domestic rivalry and the creation and persistence of competitive advantage in an industry." He further notes that "domestic rivalry is superior to rivalry with foreign competitors if one recognizes that improvement and innovation, rather than mere static efficiency, are the essential ingredients of competitive advantage in an industry."

According to Porter's model, competitive advantage of countries depend on their capability to make the following evolutionary shifts in their industrial profile:

factor-driven $\longrightarrow$ investment-driven $\longrightarrow$ innovation-driven.

In a recent paper, Rappaport and Halevi (1991) endorse this view that innovation is the prime determiner of success in the computer industry. Control over fabrication technology no longer determines superior market performance in the computer industry, according to them. What matters is "Who creates utility for users?" And utility is a function of innovation in the use of the new computers that advances in fabrication technology make available. "This means", according to them, "investment focus on software development, systems integration, marketing, and training."

To summarize, the central message is: *Innovate or perish*. And to innovate, a highly sophisticated, demand-driven, competitive domestic market is an essential prerequisite. An informed and challenging domestic market is, thus, seen to be a critical requirement for building up both export capability and innovation capability. The question to answer, then, is, "How can developing countries go about growing a viable domestic market for software?"

6. Developing a Domestic Market for Informatics Goods and Services

We have argued the importance of developing and sustaining a vibrant domestic market for informatics goods and services both from the viewpoint of meeting the development-catalyzing application needs of developing countries (Section 4), and for promoting innovation and entrepreneurial capability to compete effectively in the global software market. If we could integrate the creation of opportunities for innovation with the task of tackling the basic developmental problems, we would be able to solve two intrinsically difficult problems at the same time. We shall discuss the feasibility of such an integration through some illustrative examples in this section. However, first it must be realized that the policy-level and implementation-level issues involved in accomplishing this task are complex and solutions may well have to be very country-specific. Some general aspects of the task are discussed by Atul Wad in his paper included in Part V of this volume. As he notes there: "for the software industry the challenge is simple to describe and difficult to accomplish: identify where domestic firms can gain the most from software applications; find the relevant software technology wherever it is; adapt it to the local needs and deliver it speedily and effectively to the users."

Deploying software technology to support basic needs applications pose additional difficulties. Problems concerned with the feeding, clothing, educating, and otherwise caring of the many millions of people who are outside the market economy becomes necessarily the responsibility of the government. If technology is to be deployed in meaningful and innovative ways to tackle these problems, the innovators must be able to cut through the inertia and rigidities of the governmental system at various levels. It is in general not easy to get the governmental machinery to think and function along new lines. The issues one is confronted with here are not just economic or technological, but in a fundamental sense, social and cultural. Again, solutions to tackle such issues would have to be country-specific.

Subject to the above caveats, we shall briefly look at three application areas in this Section, which offer potentials for innovation and entrepreneurship.

6.1 Informatics Supports to Crafts

Bhalla et al. (1984) is a compilation of several case-studies illustrating the possibility of effectively blending new technologies with traditional economic activity. One of the "new technologies" figuring prominently in this compilation is, of course, microelectronics technology. As Bhalla *et al.* point out the introduction of a new technology in a traditional socio-economic environment, which could have three kinds of effects: (1) the new technology could completely sweep away the traditional occupations leading to a 'disintegration' of the traditional socio-economic framework; (2) the new technology may remain neutral and have no impact on the traditional framework; or lastly, (3) the old and the new technologies could be blended in a constructive way to revitalize and economically upgrade the former.

As Bhalla *et al* note: "If traditional production can be upgraded by a marriage with newly emerging technologies, while still maintaining much of the substance and form of the older methods, gains in efficiency and competitiveness can be achieved while preserving existing human and physical resources." (p.24) Some of the traditional economic activities in developing countries, which are potential candidates for such blending, are to be found in the agricultural and rural industry sector, the informal urban sector and among small- and medium-sized urban enterprises.

The case studies included in Bhalla *et al.*'s book of successful blending of microelectronics technology with traditional activities span a wide spectrum and include applications in agricultural management, livestock development management, biogas plant management, and so on. One of the most interesting set of case studies discussed in the book relates to the successful blending of the old and the new technologies in the textile sector. Of the several applications of microelectronics technology in this sector discussed, one concerned with upgrading the crafts skills of weavers is not only novel but seeks to tackle successfully a crucial problem one faces in the upgrading of traditional crafts skills. It is this. In order to modernize traditional crafts skills and make them more productive and marketable, the craftspersons must be endowed with modern tools, and more importantly, with modern design capabilities. In the example discussed in the book, an effort has been made to retain weaving as a cottage industry, but to underpin it with sophisticated computer-assisted techniques.

A firm located in California (AVL Looms), has designed and is manufacturing a production handloom that can be operated under computer control. It is possible to design new fabric patterns using a personal computer, and then control the dobby head to reproduce that pattern on the loom. The computer can store the pattern for future use. "All this adds to the productivity of the worker, the versatility of the loom, and the quality and marketability of the end product." (p.93). "Users of AVL Looms include home and studio weavers, textile design firms, schools and cottage industries. .... One of the advantages of AVL Looms is that even beginners are able to produce fabrics of professional quality within a few hours. .... Almost from the beginning, home-weaving on the loom can become a source of income ...." (p.94).

Clearly, analogous possibilities exist to upgrade skills in other traditional crafts (e.g., jewellery-making, wood and metal working, tailoring, etc.) through the use of modern tools and appropriate computer support.

6.2 Informatics Support to Vocational Education and Training

In the educational sector, one normally tends to associate high technology with tertiary and advanced education in the case of developing countries. However, the potentials of a technology such as informatics lend themselves ideally to enrich learning environments in schools, vocational training centres, and polytechnics. It is important, especially in the case of developing countries, to train the trainers first before addressing the students directly. It

would be logical to first upgrade teacher-training institutes through information technology (IT) and also to create IT-based environments in schools to enable teachers to prepare classroom teaching material incorporating text, graphics, pictures, maps, etc. The technology of desk-top publishing is moving in such a way that soon colour hardcopy-making devices should become available at affordable prices.

There is also a good case to be made for linking schools into a national network of an "invisible college" by stages, that would progressively link up persons actively involved in teaching *at all levels*. School teachers, at least the more innovative and motivated ones, would find themselves through this means as part of the main intellectual stream of the country. Access to resource material available at the higher educational strata (e.g., library reference material) would also be open to them in this way.

Analogously, IT can be effectively used to train teachers of the vocational training institutions and polytechnics. In the latter case, multimedia technology can be effectively used to enrich the learning environment of students.

Multimedia, expert systems, intelligent tutoring systems, networking and computer graphics, are some of the software technologies that have immediate applicability in the education and training sector. Where developing countries have to cope with multiple languages and multiple scripts, IT can lend a helping hand to make teaching and training more effective. If imaginatively promoted and developed, the domestic software market in the educational and training sector in developing countries should become a major one. Since education and training are intrinsically culture-dependent, domestically developed software in this sector should in principle have an edge over imported software — especially those imported from developed countries.

6.3 Informatics Support for the Small - and Medium-scale sector

In trying to develop a viable domestic market for informatics goods and services in developing countries, a critical task is to improve the information consciousness and increase information utilization in the small- and medium scale sectors — both in manufacturing and services. There are several reasons for investing money and effort in enlarging informatics activities in these sectors. One obvious reason is that in most developing countries the major part of business and trade transactions take place in these sectors. A second, and more important, reason from the developmental viewpoint, has been well-summarized by Botelho (1991): "The emerging conceptual framework sees SMES (small and medium enterprises) as more economically efficient than their large-scale counterparts and more attuned with integrated socio-economic development objectives of promoting employment and local capacity building. Furthermore, SMES provide a breeding ground for indigenous entrepreneurs."

A World Bank Research study on small manufacturing enterprises (Little *et al.* 1987) supports these views: "When measured by employment size, firms in the medium-size range of 50-200 workers have the highest capital productivity, and total factor productivity, in most industries in all the countries examined." (p.313).

A recent census of retail outlets in 195 small towns and 500 villages, carried out by a market research organization in India (ORG 1991), estimated their number as more than 3 million, or 5.7 shops per thousand population. The annual turnover of all the urban retailers was estimated to be Rs.110 billion (approximately US$ 7 billion at the exchange rate prevailing then). Nearly half the urban outlets — 46 per cent — were found to be grocery stores; 5.6 per cent were chemists shops; and 3 per cent were bakeries or confectionery shops. Even if a reasonable percentage of the chemists shops and bakeries were to be made informatics conscious and persuaded to underpin their activities with informatics technology (IT), the market for informatics goods and services will increase significantly in India. Analogous possibilities should exist in many other developing countries.

However, before a mass market can be developed for IT in developing countries, the environment in which small- and medium-scale business and trade transactions take place must be analyzed and understood. It is important to take into account both the physical and cultural aspects of the environment. Most of these transactions tend to take place outside organized office environments. Even a laptop computer may be unsuitable for easy use in these environments. Culturally the most acceptable human-machine interface may not be a keyboard. An electronic notepad with a stylus may be a more user-friendly interface. We need a simple device, capable of being held in the hand, battery-operated, inexpensive, which can accept handwritten characters/numerals/other notational symbols, and with enough intelligence built-in to cope with day-to-day transactions. For larger transactions and bigger shops, facilities must be available to cumulate the outcome of daily transactions, to compute monthly accounts, and other business management-related summaries and projections.

There are several lessons to learn from this discussion. First, there are deep-rooted cultural constraints that technology must adapt to in order to service the mass sector in developing countries. Secondly, shaping of technology to suit cultural norms almost always requires the deployment of leading-edge technology. Here is a state of affairs made to order for innovative entrepreneurship.

7. Conclusion: A Summary of Initiatives Needed

The principal points made in this chapter can be summarized as follows: software development is an engineering activity and so the software industry must be organized and run as an engineering industry. However, this industry differs from other familiar engineering industries in being design-oriented and not manufacturing-oriented. The design activity is becoming increasingly dependent on specialized design tools. The management of large

software projects also requires the assistance of software project management tools. Structuring support environments of this kind and training software professionals to function efficiently in these environments demand capital-intensive investments. Since such investments are prerequisites to success in a highly competitive global market, developing countries in general, should not try to promote a predominantly export-oriented software industry. It is much more beneficial, both in the short- and long-term, to develop a demanding and informed domestic market for software and support the development of a domestic software industry to cater to the domestic needs and demands.

Software is a vital component of information technology (IT) and IT is a pervasive and enabling technology with extensive potentials to modernize all aspects of life in developing countries (DCs). A strategic objective of developing countries should, therefore, be to promote software industry to provide software goods and services, not only to improve productivity, the quality of manufacturing and service industries and manpower skills, but also to come to grips with problems of poverty and resource limitations. By increasing operational efficiency through the use of IT, imports can be cut down in many sectors: the energy and transport sectors are two prime examples. Balance of payments can be improved not only through direct exports, but indirectly by cutting down on imports.

To build up and sustain a viable software industry in a developing country, closely coordinated actions are needed on the part of the government and other end-users, aid-giving agencies and the software industry professionals. Elsewhere in this volume, the initiatives that UNIDO has been taking to promote software as an industry in developing countries are discussed in detail. Here we shall briefly list some additional suggestions. See also UNDP (1991) for elaborations of these and other related suggestions.

7.1 What Governments and Government Agencies can do

Government departments and government agencies must become information conscious. While there is general realization that planning must be based on information, it is not always realized that information is even more important to implement approved plans cost-effectively. Cost overruns, time overruns, resource wastage and a variety of related ills associated with most plan implementations can ultimately be traced to inadequate availability of information and inefficient management of available information. Effective development of IT is indispensable to rectify these ills and software is the most crucial component of IT in this context.

Government organizations must realize that software development is a professionally demanding task — and it is expensive. They should be willing to pay for software products and services. This requires awareness about what software is, what is involved in software development and, most importantly, the understanding that without appropriate and high-quality software computer hardware is useless.

Government organizations, in so far as they are bound to be major consumers of software products and services, must acquire the knowledge and competence to draw up specifications for the software and services needed, tender for these products and services in a competitive market, evaluate proposals received, and monitor and enforce good software engineering practices on the part of the selected suppliers.

The government must, finally, play a major promotional role in the development of a healthy, competitive, high-quality professional domestic market for software products and services. Education and training in appropriate professional knowledge and skills; the provision of suitable subsidies; fiscal incentives and venture capital; enforcing legal protection to intellectual property rights; assisting in the advertisement of internal market strengths and accomplishments; promoting software export and similar efforts must continue to be the primary responsibility of the government.

7.2 What the Aid-giving Agencies can do

Aid-giving agencies (United Nations agencies and others) can play major and effective roles to help developing countries build up viable production capabilities for software goods and services. Wherever IT inputs are needed as part of externally-aided project implementations, the first effort must be to purchase the needed products and services from local vendors. If the ultimate objective of external aid is to assist the local development of technological skills and competence, it is counterproductive to deny learning opportunities to acquire such skills and competence to local groups.

Promoting regional cooperation among developing countries is very valuable. Equally valuable is the identification of successful institutional models that could be transported from one developing country to another. Creation of appropriate forums for exchange of information and experience among developing countries should yield rewards in the short, as well as long term.

7.3 What the Software Industry Professionals can do

Two major issues that software industry professionals must be concerned with are:

1. Distinguishing between trading in software and building up a genuinely creative, innovative and professionally managed software industry; and
2. Providing software products and services that genuinely cater to the needs of the end-users.

The industry's concern must be to look at and understand the users' problems from the users' viewpoint, and not just from the marketing viewpoint. Very often users need assistance in the analysis and articulation of their real problems. In most cases, the solutions, to be really cost-effective, must be implemented in phases. The functionalities must evolve and grow in sophistication as the end-user organization adapts to the technology and learns to use it to the best advantage of the organization. All these require an industry-customer

relationship that is interactive, creative and enduring. It is worth noting that all these points remain valid whether what is involved is system integration, custom software development, or package software sale. For, even in the case of an off-the-shelf sale, much after sales service by way of training, customization and continued assistance would be needed.

To play this kind of a role, the software industry must be driven by genuine professional concerns for technology, tools, design and development. Quality assurance and pricing are two critical factors that can either make or break the growth of a healthy and informed market. The concern of a professionally-oriented industry must be technology-related long-term growth of the market, and not short-term, high-profit, trade-oriented transactions.

Considerations of technology-related long-term growth immediately have implications for manpower training, career development and interactions with R & D and academic communities. The industry-academia interactions must be consciously and viably built up to mutually yield beneficial results.

Industry associations, through the use of newsletters and other means, must publicize success stories as well as project opportunities. Publicizing success stories is important to educate end-users on the value of software. This is a basic prerequisite to developing a viable domestic market for software.

References

A. Bhalla, D. James, Y. Stevens (1984): *Blending New and Traditional Technologies: Case Studies*, Tycooly International Pub. Ltd., Dublin, Ireland.

A.J.J. Botelho (1991): "A Strategy for the diffusion of public domain software in sub-Saharan Africa," UNIDO, Vienna, Austria.

Dimitris N. Chorafas (1986): *Fourth and Fifth Generation Languages* (Volumes I and II), McGraw-Hill , 1986.

C.M. Correa (1990): "Software Industry: An Opportunity for Latin America?"; *World Development*, Vol.18, No.11, 1587-1598.

Datamation (1980): "Picking and perfecting the packages"; December 1980, pp. 139-148.

F.C. Kohli (1991): "Software: A Process"; in *Software: An Engineering Industry — Need for developing a Domestic Market*, Proc. UNDP Seminar, New Delhi, India, 6 August 1991.

I.M.D. Little, D. Mazumdar, J.M. Page Jr. (1987): *Small Manufacturing Enterprises*: A World Bank Research Publication, Oxford University Press, New York, USA.

James Martin with Joe Leben (1986): *Fourth-Generation Languages* (Volumes I, II and III), Prentice-Hall, 1986.

R. Narasimhan (1984): *Guidelines for Software Development in Developing Countries*; UNIDO/IS.439, Vienna, Austria.

R. Narasimhan (1992): "Is globalization the answer to our problems? The case of Indian Software industry"; CMC National Fellowship Lecture.

OECD (1989): *The Internationalization of Software and Computer Services;* OECD, Paris, France.

M.E. Porter (1990): *The Comparative Advantage of Nations*; Macmillan

A.S. Rappaport and S. Halevi (1991): "The Computerless Computer company"; *Harvard Business Review*; July-August 1991, pp. 69-80.

R. Schware (1989): "The world software industry and software engineering: opportunities and constraints for NIEs"; World Bank Technical Paper No.104.

Alex Varsegi (1990): *Mainframe High Productivity Tools of the 90's*, John Wiley & Sons, 1990.

S. Watanabe (1989): "International division of labour in the software industry: employment and income potentials for the third world", World Employment Programme Research, Working Paper, ILO, Geneva, Switzerland.

L. Welke (1980): "The origins of software"; *Datamation*, December 1980, pp. 127-130.

UNDP (1991): *Software: An Engineering Industry - Need for Developing a Domestic Market*; Proc. Seminar, New Delhi, India, 6 August 1991 (available from NCST, Bombay, India).

II

Software: An Engineering Industry

Software Engineering: A Survey*

Wladyslaw M. Turski**

The term "software engineering" was coined at the same time and at the same conference that brought into the open the deep concern with the growing software crisis. The conference was held in 1968. The chief symptoms of the software crisis were: software unreliability, delays in meeting promised delivery time for software systems, difficulties in achieving desired functionality and performance of software, complexity of software systems and their resistance to modification attempts, shortage of skilled programmers and – above all – the alarming cost of software development and maintenance.

The last fifteen years saw a dramatic decrease in the cost of all imaginable units of raw computing power: the dollar-per-KB and dollar-per-MIPS measures are falling in absolute terms, let alone discounted for inflation. Parallel to this trend, although running in the opposite direction, is a very rapid increase in hardware capabilities and availability. A fixed amount of money buys not only much more hardware today than 15 years ago, it also buys a much more sophisticated equipment. Software difficulties, then, seen as a bottleneck in computer applications, have become the most important limiting factor today. For all practical purposes, the software costs are the foremost consideration when a new application system is contemplated. The policy of buying hardware to run available software systems has become almost the industry rule. The accumulated investment in software is already staggering and grows world-wide by some 10^{10} - 10^{11} US$ per year.

It should therefore come as no surprise that a very considerable effort is being put into improved methods of software design, implementation and maintenance. Qualified programmers being scarce the world over, assorted software tools, increasing the programmers productivity by automation of the more routine aspects of their work, are considered a very promising means of alleviating the software crisis.

In this report we shall survey the main directions of software engineering, a discipline for producing better software more economically. (It is important to consider both aspects simultaneously: making software cheaper at the expense of its quality is just as absurd as improving its quality at a disproportionate increase in cost).

* UNIDO/IS.446, February 1984

** Professor of Computer Science, Warsaw University, Poland.

The Change in Approach

Arguably the most important change in the whole software scene over the last 15 years is the emergence of consensus on the issue of software correctness. It is by now universally accepted that correctness is the main criterion of software quality: no matter how good a piece of software is in all other respects (such as efficiency or robustness), if it is incorrect its value is nil. Without an accepted notion of what constitutes the correctness of software, the insistence on software being first of all "correct" would be meaningless.

A very useful notion of software correctness has been found in the logical notion of satisfaction that may exist between two formal systems. Roughly speaking, a system S satisfies system T if whatever follows from the system T is a fact in system S. In more rigorous terms, we say that S satisfies T if there exists an interpretation I: T - S such that to each statement t deemed true in T the interpretation I assigns a provably true statement s in S: (t = true in T) implies (s = I(t) = true in S). In practical terms, T is what constitutes a specification, S— the software.

For example, the statement t may say that SORT (x) is the sequence X rearranged so that its members are put in ascending order. Statement s may take a form CALL PROG SHELL (INPUT(A)). If now the interpretation I is such that PROC SHELL is the name of the Shell-sort routine, CALL denotes an invocation of a routine, A is the name of a file, INPUT is an operation delivering the file listed as its parameter, then all that is needed to establish that s satisfies t is to prove, from the particular description of Shell-sort, that indeed its execution delivers the sorted version of its input parameter.

Two observations are in order :

- — Using the outlined approach we assume that the specification correctly reflects the user's requirements.
- — It is a matter of formal calculations to establish if a given software satisfies the given specification.

The first of these observations clearly indicates that the burden of somehow verifying whether or not the software meets the application needs is shifted from programming to specification analysis. The second one presents in a nutshell the methodological advance, which is the cornerstone of software engineering: given the specifications, the correctness of remaining parts of the design and implementation process becomes a calculable question.

We shall return to the specification issue later. Now, we shall consider the significance of correctness calculability.

First attempts to exploit the notion of calculable correctness concentrated around programme verification, i.e. around a process that would take a specification and a programme, and — based on this information alone — would attempt to calculate if the programme is correct. This approach has had a limited success only: the amount of formal calculations

involved was formidable even in the case of pretty small programmes. For large programmes it became prohibitive, even if human intervention was allowed to speed up some of the calculations.

Soon it transpired that a much better policy to exploit the notion of calculable correctness is to devise such programming techniquesthat would guarantee programme correctness by virtue of the very construction process. Thus, methods of building correct programmes from specifications started to appear.

Specification-based Methods of Programming

Common to these methods is the view of specification as the most abstract description of all desired properties of the desired software. (By the "most abstract" here we mean "free of all unnecessary detail.") The software design and implementation is seen as a process of transforming such abstract description, by adding necessary details, into a programme which is to preserve all properties contained in the specification. (Thus the main difference between the specification and the corresponding programme is that of detail: the specification is free of machine-oriented details but, of course, contains all application-oriented ones.)

Methods that pursue this approach are known as top-down design methodologies, a name that refers to the fact that — after the software is successfully designed and implemented — the history of the construction process is not unlike a pyramid, with the original specification occupying its summit and the working version of the implemented software being its base. When one ascends this pyramid (moving as it were in the direction opposite to that which the designer took), one sheds the implementation and design details until the refined, most abstract summit — the specification — is reached.

Basically, the top-down programming methodology consists in repeated application of the following procedure:

— Given a problem P, is it possible to express its solution in a reasonably concise fashion using primitive notions of the linguistic level at which we want to programme? If yes, write the programme, if not, invent notions P1, ..., Pn such that

- each of the notions P1,..., Pn is well-specified.
- using these notions according to their specification it is possible to write a satisfactory programme for problem P.

— Consider each of the notions P1,..., Pn in turn as a new problem and repeat the procedure.

This procedure continues until all invented notions are implemented in terms of primitives of the given programming level.

The above given brief description of the top-down design and implementation methodology introduces two important techniques: that of structured programming (or structured decomposition) and that of stepwise refinement. We rely on the first one when we decompose problem P into problems P1, .., Pn, and on the other one when we consider each of the P1,...Pn as a new problem in itself, to be solved by the same method.

Both techniques rely on sound mathematical principles, which guarantee their correctness if certain rules of decomposition and refinement are observed. Consideration of the mathematical foundations of structured programming and stepwise refinement have led to new concepts in programming languages, such as ADA, PASCAL or MODULA. These modern programming languages are designed explicitly so as to facilitate and even in some instances enforce a disciplined use of these techniques. It should be observed that while no programming language *per se* ever solves any software design problem, the use of a tool influences the way in which its user works. Programming languages of yesteryears, FORTRAN, COBOL and BASIC, lack the mechanisms to support structured programming and stepwise refinement. In such languages it is virtually impossible and certainly very awkward to use the techniques that emerged as the most common tool of modern software engineering.

Approaches to Specification of Software

It has been firmly established that the early stages of software system design are crucial for the eventual system usefulness. This observation is, of course, directly related to the fact that it is precisely the specification, which in the final count is taken as the frame of reference in which software correctness is established. Thus, all design/implementation techniques respecting the notion of correctness cannot but preserve any specification errors. Consequently, such errors become apparent only after the software has been implemented and are thus very expensive to correct. This clearly underscores the need to verify specifications, both from the point of view of their implementability and from the point of view of their relevance for the intended application.

The verification of specifications with respect to their relevance poses a very subtle question of translation between (often fuzzy) intentions of the eventual user and (necessarily formal) expression of specifications. Many techniques have been proposed specifically oriented towards easing of this task. In essence, these techniques assume that the prime author of the specifications is the eventual user, and — by providing a somewhat restrictive means of expression — force him to express his intentions in a form that can be easily manipulated by formal methods. Often such techniques depend on graphic conventions (e.g. SADT), whereby the user/author is pressed to describe his ideas in the form of pre-designed and partially labelled diagrams. Freely added, user-invented labels express his intentions originally just by their mnemonic significance. Gradually, as the specifier is asked to complete

more detailed diagrams, the pre-designed structural dependencies between diagrams are explored by a "hidden" analyser, which brings to the open all inconsistencies and many instances of design incompleteness.

An entirely different approach to specification writing can be exemplified by specification languages (such as CLEAR). In this approach, the recognition that a specification is in fact a formal theory of an application domain is made into the main tenet and main mental tool as well. Specification languages provide means for relatively easy-to-read description of theories and — more significantly — for combining thus described theories into larger ones. For example, given a formulation of a theory of optimality (for instance by means of linear programming principles) and a formulation of a theory of control of a chemical process, their combination will yield a formulation of a theory of optimal control of this process. (Naturally, not any two theories can be combined, it is up to the specification language designers to make sure that any combination expressible in the language "makes sense" and that absurd combinations would be inexpressible; exactly as in safe programming language, it is impossible to execute sin(true) instruction).

When many useful application domain concepts are captured by corresponding theories, a specification language may indeed be very useful: the specification of a concrete system may be obtained by a combination of "library theories" with some specific expressions written just for this system. The main advantage of this approach, apart from economy of design, is in the increased safety: library theories are known to be safe and the language includes many safety measures, which make it unlikely that nonsensical combinations would be expressed by mistake.

Both graphic and linguistic approaches to formal specification writing are well founded in deep theoretical research into such issuesas abstract data types, algebraic theories, theory of models, etc. The same kind of foundations are used by a number of software description techniques, such as VDM, commonly used for unambiguous definition of large software products such as semantics of new programming languages (CHILL, ADA) or special software systems (CICS).

Software Life-cycle

A software system written for a particular application can seldom be used for an extended period without undergoing a number of changes. Among the many causes that necessitate software changes one can list the following:

a) The nature of the application itself changes (e.g. for a banking application, the introduction of EFT facility changes dramatically the accounting procedures).

b) New hardware elements are added to the system and need to be incorporated into the class of devices supported by the software (e.g. colour screens are introduced into a system that used monochromatic screens only: a new "dimension" must be added to all output and, perhaps, input functions).

c) An existing piece of software is transferred to another environment, or the environment itself changes (e.g. better educated operators are hired, for whom the existing input procedures are too dull; new input procedures are required, which would make the operators' task more appealing to new staff).

d) The scope of the application is enlarged (e.g. in a hospital computer system the intensive-care unit computer services are to be linked to a previously separate medical record system).

e) The requirements placed on the existing system are changed (e.g. the air-traffic control system must be modified when the airport it serves starts accepting faster jets and thus the decision time must be reduced to accommodate the faster traffic).

The list of causes can no doubt be extended. Even this incomplete list is sufficient to draw the unavoidable conclusion that changes in a live system are indeed necessary, quite apart from any remedial changes due to its detected shortcomings and internally motivated software improvements.

An unfortunate tradition lumps under the title "software maintenance" all those activities related to software changes that occur after the original system has been successfully installed. It is important to remember, therefore, that software maintenance is needed not because software deteriorates in use (there is, of course, no wear and tear of software), but because its use for a dynamically changing application will be diminished unless the software is modified.

In fact, the usual presentation of a software life-cycle contains four major phases: (a) Conception (b) Design (c) Implementation (d) Maintenance

Largely due to the fact that "maintenance" comprises future modifications, its share in the total expenditure is very large. In many cases the maintenance costs constitute more than three quarters of the total investment in a software system. (This is a very important observation: a client buying a moderately-sized software system for, say, $100,000 should not be surprised that the maintenance costs over the next few years will run up to $ 300,000. If he is not prepared to pay this "extra", almost certainly he will find himself tied to an ever decreasing useful piece of software. If he tries to economize, e.g. by assigning junior staff to maintenance activities, he may face a total disaster — the system may become hopelessly entangled and virtually useless.)

The main reason for the high cost of software maintenance is the fact (abundantly confirmed by many a post-mortem analysis), that the complexity of software grows very rapidly with every change made to the original version, unless a conscious redesign effort (also expensive) is made to reduce the complexity. Thus, each subsequent change is harder to make and is more likely to introduce — in addition to the desired ones — many unforseen and often unpleasant effects.

These observations lead to two inter-related software engineering problems :

a) How to design software in such a way that it would be relatively easy to modify?

b) How to modify an existing software so as not to increase its complexity more than absolutely necessary?

The first of these issues is answered by modular and hierarchical design. The second — by controlled backtracking techniques — is greatly facilitated by well-designed programming support environments.

Modular Design

A very general engineering principle calls for the final product to be assembled from easily replaceable parts. Thus a bicycle consists of a frame, two wheels, pedals, chain, etc. If the bicycle malfunctions, the case of the trouble can usually be traced to one of these parts, and the faulty part can be replaced by another of the same type or, indeed, by any similar part that fits. It is not unusual for the bicycle to have wheels of different make to the frame or even two different pedals. As long as the parts satisfy certain externally specified interface requirements, their internal construction is of secondary importance. Thanks to this principle, we may put snow tyres on our cars, thus obtaining a vehicle with rather different driving parameters without actually having to change the engine or steering wheel.

This general engineering principle translates in software design into a requirement according to which any software product should be built from relatively independent modules. Each module meets its specifications if it is a correct module; a module specification is all that is externally known about the module. The functionality of the whole system obtains from the interaction of modules, the interaction itself being fully determined by modular interfaces.

When designing a piece of software, one decomposes the design into a number of relatively independent units — modules. Having listed the external properties of each module, i.e. having formulated each module specification, one can prove the appropriateness of the decomposition by proving that the required properties of the whole indeed follow from the postulated properties of modules. This being established, each module in turn may be considered a new design problem. Thus, allowing for hierarchy of modules, we see a complete analogy with the stepwise refinement technique, although this time the technique is expressed in terms of structured components of programmes.

A well modularized program can now be relatively easily changed by replacement of one module by another, the fresh module sharing with the old one its interface specification, while differing in secondary considerations, i.e. in those that are not covered by the specification relied upon in the overall design. For instance, if we want to adapt our software to exploit the potentials offered by a new output device, we can concentrate on the output module in which all relevant aspects of the given piece of software should be encapsulated.

Naturally enough, not any haphazard hacking of the design into pieces can be considered a proper modularization. Sound engineering principles of modular design have been formulated, which facilitates making correct modularization decisions. The same principles ensure that a majority of errors can be localized within a module, thus the repair activities may usually be limited to a single module

Since a module can be replaced by any other module provided their functional specifications and interface characteristics match, there is a huge incentive to plan libraries of interchangeable modules, rather like a Mechano set, from which a variety of software products could be rapidly constructed. Modules that could be used in many different programmes are known as reusable ones, and all successful software houses possess substantial libraries of reusable modules from which a major part of any system within the specialization area of the house can be constructed.

Some modern programming languages, notably ADA and MODULA, actively encourage modular programming. ADA, for example, was explicitly designed to permit independent compilation of modules, so that libraries of resuable, precompiled modules may be accumulated.

A number of design methods have been proposed that are based on the modular programming principle. Some of these are commercially available as kits consisting of a large number of tools, i.e. special programs that assist programmers in their work on software design and implementation. In general, such commercially available methods concentrate on a chosen guiding principle for modularization (e.g. data flow, calling hierarchy or input/output transformation), which suggests a particular approach to structuring the design. Identified modules are named and their main characteristics are specified. Then, the use of supplied tools enables one to display the emerging design in a coherent way and — more importantly — check the consistency of the design by verifying that the specified properties of the modules placed in their respective structural positions indeed correspond to each other. Thus, for instance, if it follows from the design structure that a module M imports data named x, it is possible to check that there is at least one other module that exports the so named data. Having identified such a module, say N, it is possible to check if modules M and N are structurally related in such a way that data transfer between them is allowed. Similarly, it may be checked if data z, generated by a module K, is imported by any other module or presented as a system output.

Most available tool-assisted modularization methods allow many such consistency checks to be run at several levels of detail, thus permitting tthe verification of at least some aspects of the design before any detailed programming (on intramodular level) is done.

Another useful extension of modular programming techniques consists in substituting module surrogates for not yet implemented modules. Thus, as the implementation progresses, one can animate the complete system even if only a part of its modules is actually coded: the remaining modules being replaced by surrogates, the whole system may be made

to perform. Such animated execution allows one to check some external properties of the system long before its implementation is completed and, therefore, to avoid at least some unpleasant surprises and — perhaps — to introduce design modifications before they become prohibitively expensive to carry out.

Software Modification

Assuming that we have a well designed and correctly implemented piece of software, any subsequent modification should start with a clearly specified request for modification. If the history of the design is preserved (and again there are special software tools constructed specifically for the purpose of storing the software design history in a manageable form), it is possible to identify the design step at which the decisions contrary to the requested modification were made. (Such a step must always be there since otherwise the request could be met by the existing software and thus no real change would in fact be requested, although the request may still have led to some programming, e.g. of an add-on extension of the available software.) As soon as the pertinent design step is identified we know that all preceding steps can be safely preserved in the new design, i.e. in the design aimed at satisfying the considered change request. If the subsequent steps of the old design are discarded, the incorporation of the requested modification may be viewed as a continuation of the preserved part of the old design by a suitable sequence of the new design steps.

If there are many such modification requests, we are soon faced with a "forest" of designs — from each design step at which some change has been incorporated a new branch is started. A suitable "navigation" tool is needed if the programmer is to be able to freely traverse the design "forest" and collect design steps along each particular branch. Again, such tools are commercially available.

The main objective of controlling the software modification may be stated as a stability problem for software development: how to achieve the situation in which small changes in specification could be accommodated by small changes in the implemented software.

No general solution for this problem is known (and, according to many experts, such a general solution may never be discovered). There exist, however, some design/implementation techniques which admit a partial solution of the stability problem.

For instance, if an arbitrary decision is made every time (selecting a particular branch), all feasible although discarded decisions are duly recorded and preserved in the design database, subsequent design steps may include a "what if" analysis, forcing the programmer not to do anything detrimental to the implementation of the rejected options. Or — at the very least — to clearly mark subsequent decisions with comments informing about such past options, which from now on become unfeasible. If the design history is decorated with such comments, a change request may be run against the tree of rejected alternatives, yielding not only the level to which the design is to be backtracked, but also informing about the point at which the incorporation of the requested change became unfeasible in the present implementation. Analysis of this information permits the rough guessing of the amount of

redesign effort needed to make the change. Changes that would require too much effort may then be rejected, or if undertaken, may be explicitly marked as difficult-to-implement and therefore expensive. It should be stressed once more that a proper management of software modification and change requests is probably the most important aspect of software engineering, as it is this phase of the software life-cycle in which the lion's share of the total expense is incurred. The general approach outlined above may be viable only if during its entire life-cycle a software system is supported by a comprehensive design and development documentation in which all versions and mutations are recorded in a manner allowing for a relatively easy restoration of arbitrary past states. It goes without saying that if such a support is to be of real assistance to programmers, it must provide the relevant information in a form that permits machine-assisted manipulation of a large number of variants. Thus, again, we are led to consider the importance of software tools.

Software Tools

There are several varieties of software tools more or less available on the market, many more tools are the property of software houses, which use them for their internal purposes.

Probably the most common among the commercially available tools are all sorts of editors, i.e. program that facilitate program composition and program text manipulation at a programmer's work station (e.g. on a VDU). The editors range from pretty simple text manipulators to quite sophisticated, programming-language oriented structured editors that in addition to the facilities provided by plain text editors include an active mode of assistance. In the latter mode, structured editors prompt the programmer as to proper instruction formats, check for syntactic completeness of phrases, warn against simple context-detectible errors, etc. Nearly all varieties of programming editors take care of simple yet laborious editing operations, such as textual substitutions, systematic renaming of programming objects, etc. Many newer programming editors are geared to exploit display facilities offered by modern terminals, e.g. by providing the so-called multiwindow option, whereby a programmer may divide the screen into several independent "windows", conducting in each of them a separate program (or program part) development. Thus for example, a programmer may develop the main program in one window, a subroutine in another and an input/output handler in yet another. A fourth window may be used for display of pertinent parameters, such as the number of lines of code already generated, memory maps, etc. Each of the windows may be zoomed in, therefore providing a more detailed view of a particular feature, contents of different windows may be merged, etc. A programming editor with a multiwindow facility creates a fair analogy of the programmer's desk top, with all assorted documents and scratch pads being available in an electronic form at the same time.

Another kind of commonly used programming tool is represented by program generators. For a variety of applications, working programs are sufficiently stereotyped to permit their automatic generation from a suitable chosen set of design parameters. Such program generators usually work interactively, either in a dialogue (question and answer) mode, or

by menu selection technique, whereby a programmer is shown a number of options, depending on the circumstances selects one, thus fixing a design decision, which triggers a next level menu to be displayed. At the end of a session, the totality of decisions made determines a particular application program, which is then produced ready for operation. Programs produced in this way are often a bit inefficient, but otherwise quite acceptable and can certainly be used as system prototypes. If their functional behaviour is found to be satisfactory, their performance may be improved in a number of ways, e.g. by optimizing the most frequently executed parts of the code. (Incidentally, program optimization is another task that may often be left to a suitable software tool.)

Finally, we should not forget that the ever growing body of commercially available compilers, interpreters, decision-table processors, etc. are all in fact software tools. An extremely useful addition to this class of tools are program transformation systems, which accept a program expressed in an abstract form and — guided by the programmer — perform textual transformations aimed at replacement of abstract algorithms by concrete ones, or at replacement of less efficient parts of code by more economic versions. The importance of such tools rests in the fact that program transformation systems are so designed that their action preserves the intended meaning of programs being transformed. A programmer may therefore try a number of transformations quite safely: even if he does not achieve the intended improvement, he certainly does not run a risk of losing correctness of his program.

Another class of software tools are those which gradually transform an initial design into a more and more program like text. Some tools of this class for instance, accept graphic designs in the form of interconnected, labelled boxes and represent them as equivalent linguistic structures more amenable to further textual refinement. (In addition to transformations between various levels of abstraction, such systems usually perform a number of useful consistency checks.) Nearly all software design methodologies advocated for general use rely on some tools of this class.

The most sophisticated software development tools, in addition to all previously listed facilities and standard features (such as parsers, table generators and file managers), also incorporate very advanced databases in which program versions and mutations are stored for easy reference and manipulation.

It is customary to refer to a fully developed system of software support tools as program support environment (PSE). Modern requirements for a programming language usually include a specification for a PSE. Probably the best-known of them is the APSE — intended programming support environment for ADA. In fact, it is expected that the portability of the ADA program will be achieved via functional equivalent of APSE installed in nearly all computers. In this way, not only the programs themselves could be ported, but also their PSE, which would allow for a further development of the program to be ported from one installation to another.

In addition to program design/implementation tools, a well-developed PSE includes a variety of tools for program testing and debugging, which greatly reduce time and effort needed for the pre-release servicing of software.

Non-procedural Languages and Other Advanced Concepts in Programming

Classic programming languages, COBOL, FORTRAN, PASCAL and even ADA, are based on the notion of assignment: as a result of the execution of a statement, a variable is assigned a new value. This notion, a direct descendant of the machine order "execute and store", can be rightly described as the cornerstone of the von Neumann computer architecture.

New architectural concepts, such as data flow machines, highly parallel computers and inference engines, find little use for the notion of assignment. Hence, in recent years a number of entirely different programming languages have been proposed, based on very unorthodox principles and more or less directly incorporating these architectural innovations, which seem most promising from the application point of view.

Arguably the most widely accepted of the new line of programming languages is PROLOG, a language for programming in logic. Originally intended as a tool for computational linguistics, PROLOG is rapidly becoming the main programming language for artificial intelligence, slowly replacing the old time favourite, LISP in this role.

What an assignment statement is to FORTRAN or PASCAL, a Horn clause is to PROLOG. (A Horn clause is a special form of a first order predicate calculus formula.) Its main computational advantage rests in the natural parallelism of Horn clause evaluation, while its advantage for application programming obtains from the observation that a Horn clause, may be just as easily interpreted as a statement of fact and as statement of a hypothesis to be verified based on other clauses. Close kinship of Horn clauses and relations, which provide the foundations for commonly used relational databases, is another bonus for PROLOG adherents: not only does it yield an easy interface between application programs and databases, but also makes possible database description by essentially the same means as program description.

Data flow machines seem to provide a natural evaluation mechanism for another brand of new programming languages, the so-called applicative languages, exemplified by LUCID. Hardware/software experimental systems, e.g. ALICE, are currently being built to explore new and apparently very powerful concepts of data flow and functional programming.

Finally, the Japanese Fifth Generation project, extremely influential in shaping the current research interests both in the United States and in Europe, favours a highly parallel machine architecture harnessed to a multilevel hierarchy of specialized machines: database machine topped with an inference engine (programmed in PROLOG) yields an expert knowledge base which, interfaced with very advanced input-output machines (such as graphic or visual computers and speech analysers and synthesisers), are to become intelligent computers of the 1990s.

Futuristic as such designs may seem, they do underline a new approach to computing: a merger of hardware and software design, aimed at exploitation of the potentials made available by the abundance of cheap and very efficient large chip components.

Software Engineering Management

As soon as we recognize that software writing has become the major component of the multi-billion dollar information processing industry, we must realize that this kind of industry generates its peculiar managerial problems and leads to specific managerial techniques.

The peculiarities of the software industry are quite pronounced. First of all, it is an industry almost totally independent of any raw materials and almost zero-energy consuming. Apart from software tools — themselves produced within the software industry — it does not require much investment in material terms. Its products are often classified as intangibles, and, contrary to any other industry, most of its costs are concentrated in design: the actual production costs (if one considers the reproduction of once composed programs as "production") are practically nil. This should be contrasted not only with usual industrial sectors, such as mining or manufacturing, but also with innovation-intensive industries, such as civil engineering or the drug industry. (A construction firm may redesign a bridge ten times over, the cost of nine discarded designs is negligible compared to that of actually building a bridge on site. A software firm forced to redesign a major piece of software usually comes close to bankruptcy).

In addition, nearly all resources needed for the software industry are people: highly qualified, well educated software engineers. Hence, the managerial problems in this industry are almost exclusively pure problems of the workforce management, where the workforce in question is very independent and fully aware of its own value.

To fully understand the ensuing managerial difficulties, let us consider a simple example: a software team engaged to produce a system for an application is running behind schedule. The manager decides to accelerate the rate of progress and implements this decision by hiring more programmers. Instead of the expected acceleration, the work is slowed down as the fresh programmers need to be instructed on the system in production and the only available instructors are the programmers already on the job. Thus the newly hired hands are unproductive because they are not "in" yet, while the old hands, burdened with the additional task of instructing the newcomers, become less productive. Hiring even more programmers may be disastrous. After a period, the newcomers are integrated and the work may resume in ernest (although the delay has grown considerably). Now, however, the management discovers that enlarging the team has blown up the communication problems, which grow as the square of the number of people on the project, and a sizeable portion of the total effort must be spent on overcoming the resultant communication clashes. Hiring an extra support team: secretaries, information officers and technical writers, compounds the difficulty. After another couple of months the management realize they are fighting an

uphill battle: the effective rate of system production is invariant of the work expanded; it seems to be solely determined by the original design, and after it has been approved there is little that the management can do to influence the rate of further development.

A number of managerial techniques, including the spectacularly successful "chief programmer team" approach pioneered by IBM, has been proposed and found useful. Still, the management of software projects remains a vexing problem and successful software managers are — if anything — even rarer than good programmers.

Impact of Technological Change on Software Development: Implications for the LDCs and NICs*

Robert Schware**

1. Introduction

The worldwide computer industry has been characterized since its inception by continuous innovation, improvement and rapid change. These developments have resulted in the growing importance of software — the programs that run computers and allow them to communicate with each other through data networks. Developing (LDCs) and newly industrializing countries (NICs) seeking a share in the burgeoning global information Industry are promoting the development of their software industries through a variety of policy and institutional measures. At the same time, a growing number of United States and western European software firms and non-software firms (such as aircraft and financial service providers) are beginning to develop software overseas often for foreign as well as domestic markets.

The worldwide software industry is still very much an *industrie nouvelle*. Any assessment of the worldwide software market has to rely on scanty, inconsistent and unreliable data. This makes projections of future software products and services an exceedingly risky undertaking. Nevertheless, such projections have been made, and they range from $US 70 billion to $US 180 billion by 1990.[1]

Five major forces are now promoting rapid changes in the world-wide software industry:

a) A productivity bottleneck in programming, with software and software-related support activities now accounting for the overwhelming percentage of total system costs;

b) The global battle for operating system standards;

c) The move away from single vendor solutions as the typical way for organizations to meet their information systems needs towards customized, integrated, multi-vendor hardware and software solutions;

* *Microelectronics Monitor*, April 1989, pp.47-53

** Senior Information Technology Specialist, Development Informatics Unit, Asia Technical Department, World Bank, Washington D.C., USA.

d) The increasing emphasis on software production and sales by hardware vendors, leading to increasing concentration by large- and medium-sized firms; and at the same time

e) an expansion and fragmentation of the industry resulting in a large number of independent software vendors.

Technology defined broadly as procedural methods, organizational modes and technological knowledge used to transform inputs into outputs, is becoming an increasingly important element in international competitiveness for software firms. This paper examines the importance of organization and management of the software production process for firms. It reviews some new software management practices being introduced by firms to control costs and improve the quality of both the final products and the process of developing software. Some new technologies that will affect the entire software production process and that will require a reorientation in thinking about investment and technological options are also examined.

2. Software Development Management

Software firms in developing countries and NICs alike are still struggling with inappropriate tools and methods in the software development process. This problem is due to: (a) lack of experience; (b) lack of knowledge or discipline; (c) difficulty of measuring the development effort accurately; (d) implementation of designs whose poor quality does not surface until the finished product is either tested or installed for operation; and (e) high life-cycle costs resulting from a system that was not designed for reusability or maintainability. Without established process methods and techniques, firms often solve the same problems over and over again. The average large software project continues to cost twice as much as its initial budget and is completed a year or more behind schedule; approximately 25 per cent of such projects are never completed and the remaining 75 per cent require inordinate amounts of maintenance.

A major premise in software development management is that the quality of a piece of software is governed largely by the quality of the process used to develop and maintain it. In a "mature" process, the methods, techniques and technology are used effectively and produce reasonably consistent results. Improvements in quality and productivity occur in part from automation. But in an immature software development process, unpredictable results occur; formal procedures, cost estimates and project plans are

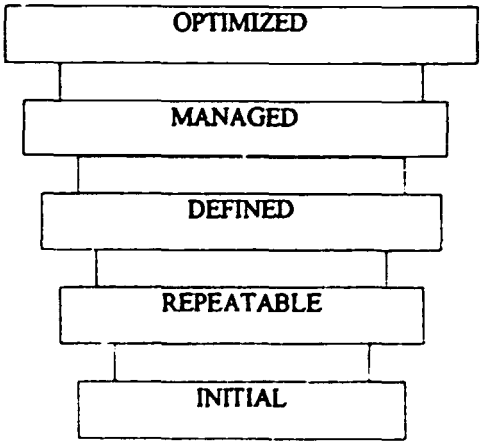


Figure 1. Software Maturity Model

lacking; and technology is used on an *ad hoc* basis. The prospects for the NICs in software development will depend increasingly on the processes carried out in the software life cycle. There is considerable middle ground between these two organizational extremes. An ideal maturity model (illustrated in figure 1) by which firms and organizations can judge the effectiveness of their software development process has five maturity levels (1) initial, (2) repeatable, (3) defined, (4) managed, and (5) optimized.[2]

The initial process level : A firm has ill-defined procedures and controls. Typically, it operates without project controls and does not integrate tools and techniques with the process. Coding and testing are dominant activities. Established procedures, if they exist, are usually abandoned in a crisis.

The repeatable process level : A firm has established basic project controls, such as project scheduling, coding standards, product assurance and change control. Mechanisms are in place for ensuring that the design team understands each software requirement. Statistics may be gathered on software code and test errors. The strength of the firm may also stem from its prior experience of doing similar work, but it may face major risks when presented with new challenges.

The defined process level : Standards and methods for technical and management activities required for software development are established at this level. These specifically include design and code reviews, training programmes and increased organizational focus on software engineering, including measuring specific tasks in the process. Some uncertainties remain about the value of the measurements, the best ones to use and the appropriate response to reviews.

The managed process level : A firm typically has a minimum set of process measurements for each stage in the software life cycle, and it conducts extensive analyses of the data gathered during reviews and tests. Automated tools and techniques are used increasingly to control and manage the development process, as well as to support data collection and analysis.

The optimized process level : Firms at this level have achieved a high degree of control over their process, have automated data gathering and typically have a method for improving and optimizing these operations. This includes identifying and replacing obsolete technologies, more sophisticated analysis of the error and cost data and the introduction of error cause analysis and prevention studies.[3]

Some firms and organizations, particularly in the United States, are beginning to use the maturity model to assess their software development process and software management. The importance of such assessments is to indicate the maturity and technological levels at which a firm is operating. More importantly, it may indicate the strong and weak areas of a firm's software development capabilities, thus identifying immediate improvement priorities, interim improvement goals and progress measures.

One implication of the maturity process model for software producers in developing countries and NICs is clear: technology and managing the process of engineering tangible products will become increasingly important. An understanding of the maturity model and how it might be of practical benefit should be a clear research priority for software firms and for various government sponsored research programmes.

2.1 Software assessment standards

Standards for software engineering will take a long time to develop. But with the rapid technological change and increasing competitiveness characteristic of the software industry, government agencies and other large purchasers of software are using new techniques to evaluate contractors' abilities to develop software according to "modern" software engineering methods. Corporations, also mostly in the United States, are using the new evaluation techniques to assess their own ability to create "critical" projects and to assess their overall competitiveness. One methodology, developed by the Software Engineering Institute of Carnegie-Mellon University for the United States Air Force, examines a contractor's capabilities in:

a) Organization and resource management, including software engineering training and the adequacy of support facilities;

b) Software engineering process and management, which includes the scope and use of conventions, formats, procedures and documentation during the various software development phases, software quality control and data management; and

c) The tools and technologies a contractor may use in the software engineering process.

Since this new methodology has become only one additional part of the criteria for procurement, it is unlikely that it will create a radical change in the contractors who are selected for software development projects but the evaluations will probably lead major software companies to manage their software development process more closely. An awareness of software producers of these evaluation methodologies is important to future software developments. Raising some of the questions presented in various methodologies may improve the software development process in software firms and to some degree enhance their competitiveness in international export markets. These methodologies are usually concerned with standards and practices of software firms in the following areas:

- organization and resource management, including quality assurance, process management, configuration control and the quality and quantity of resources;
- software engineering process and its management, including the scope, depth and completeness of the process and how it is measured, managed and improved; and
- Tools and technologies used in the software engineering process, e.g., computer tools to measure test coverage, to analyse cross-references between modules and to design and debug code.

2.2 Software metrics

The state of software metrics, or measurement, is still somewhat immature and imprecisely defined. None the less, measuring software and software development can lead to substantial benefits for reasonable costs. Short-term productivity improvements, as well as the establishment of common development environment, have been reported by companies that use software metrics. To many project managers in software companies, software metrics are a means for more accurate estimation of project milestones, as well as a useful mechanism for monitoring progress.[5] Basically, software metrics are a way of measuring the various attributes of the software development process. Attributes include the size of a program, its cost, the number of programming errors or defects, the level of difficulty or complexity of the project and the method of communications required between members of a project.

Figure 2 shows an example of using software metrics in software development in what is usually an ignored activity: documentation. Documentation is frequently a low priority activity in software development. Yet in two studies at IBM and TRW, the overall quality of software is very much a function of documentation. As the figures suggests, 25 per cent of all software defects in the two studies were in documentation delivered to customers.[6]

Many tools are available — some in the public domain — to assist in collecting and examining measurable results during the various development stages. An earlier issue of the *Microelectronics Monitor* (No. 24, 1987/IV) describes some of the available public domain software.* For example the United States National Aeronautics and Space Administration (NASA) offers more than 1,100 computer programs through its Computer Software Management and Information Center (COSMIC). A program called SOFTCOST estimates software size, implementation productivity, recommended staff level, probable location, amount of computer resources required, and amount and cost of software documentation. SOFTCOST is designed to provide project managers with a comparison between their expectations of a project's development and industry-based

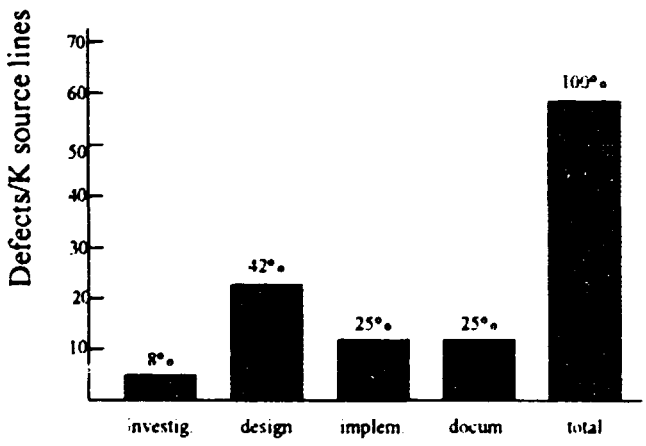


Figure 2. Introduction of Software Defects (from IBM and TRW studies)

* This paper is reproduced in Part V of this volume.

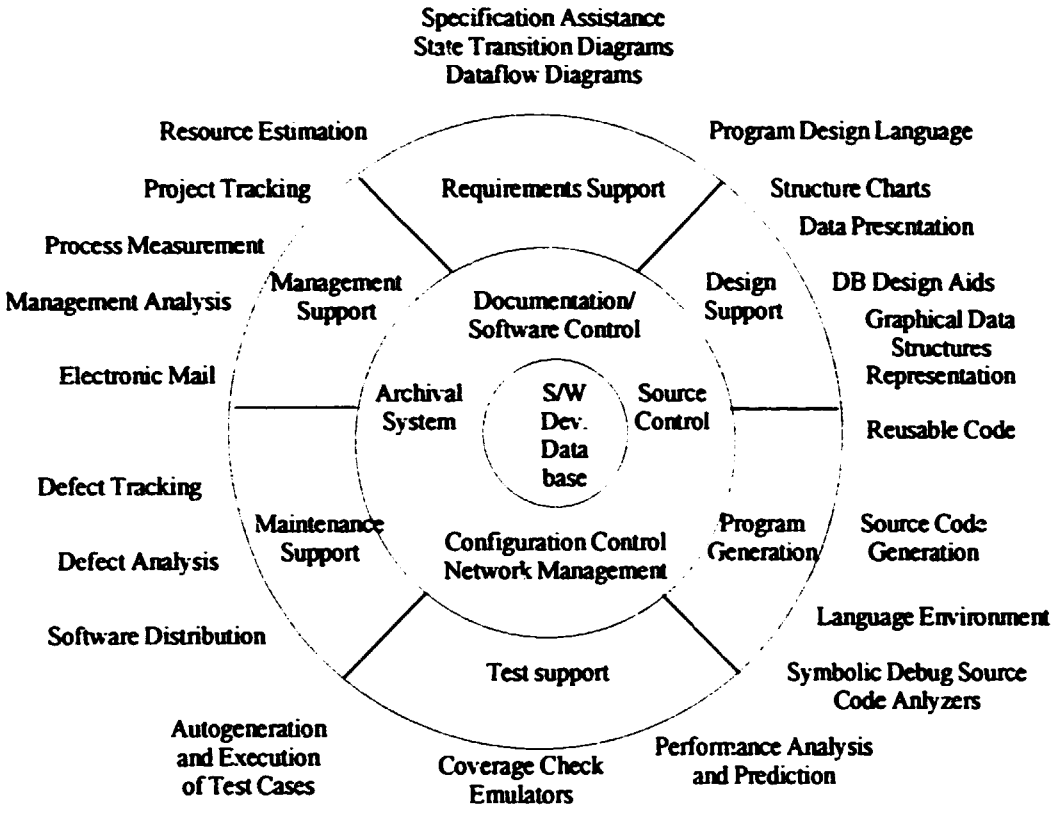


Figure 3. Hewlett-Packard Software Engineering Productivity Environment

statistical expectations of that project. The software metrics data provide an additional basis for budgeting time and effort to a project. A complete integrated set of tools does not exist. Figure 3 illustrates what Hewlett-Packard considers an ideal software development environment, which is reinforced by tools and metrics. The central database provides a common control mechanism for all important parts of a project.[7]

"Successful" integration of collection and use of software metrics into the software development process is the main objective of firms using such tools. The time factor involved in learning tools and incorporating them into the entire development process has been the primary obstacle of their implementation. For example, the history of Hewlett-Packard Software Metrics Council shows that it took roughly three years of collecting analysing data using software metrics before there were sufficient data available to show measurable trends for the entire organization.[8]

2.3 Software risk management

Several years ago, an error in the avionics software for the F 16 jet fighter instructed the plane to flip upside down whenever it crossed the equator. In separate incidents, software "bugs" have been responsible for the deaths of patients when an irradiation unit for cancer therapy generated "inappropriate" doses. More recently (1988) American Airlines lost an estimated $US 50 million in ticket revenues because its passenger reservation system indicated that aircraft had sold out of discount fares when such seats were in fact still available. The list goes on and on.

Like many fields in their early stages, *software engineering has had and continues to have its share of disasters*. Most assessments of projects with large, critical and complex software systems that "failed", particularly in banking, air traffic control, nuclear reactors, chemical plants, medical technology and defence and aerospace systems, have indicated that software-related problems would have been avoided or at least significantly reduced if there had been an explicit early concern with identifying and resolving high-risk elements. Frequently, these projects were swept along by a tide of optimistic enthusiasm during their early phases, or by enthusiasm for new software capabilities.

Risk management is an emerging discipline that provides techniques both for risk assessment (risk identification, analysis and prioritization) and risk control (risk management, planning, resolution and monitoring). To ameliorate the potential problems caused by the relative unreliability of software systems, the United States Government is increasingly requiring risk management plans of software contractors (for example, the Federal Aviation Administration's Advanced Automation System and the software for the United States Air Force Small Missile System). Software risk management is also being instituted within United States industry.

Table 1 provides a list of risk items in software projects and techniques in the software life cycle that have helped software projects avoid disasters.[9] It is theoretically conceivable that a project could face all of these risks. Some of the software risk management techniques may appear familiar to software developers, although they are usually applied in an *ad hoc* fashion. Provisions for risk management should be made in the planning process by software firms, particularly those designing large systems with a number of independently modifiable subsystems and interfaces with other systems.

3. The Impact of New Technologies

Research and development efforts in the software engineering have produced new methods that show promise for improving programmer productivity and software reliability. As the software industry gradually becomes less labour-intensive over time, the quality and availability of skilled labour will become more important than labour costs. Pressures on the quality of software labour markets will increase competition among NICs, as well as

Risk Items	Risk Management Techniques
1. Personnel shortfalls	Staffing with top talent; job matching; team building; morale building; cross-training; prescheduling key people
2. Unrealistic schedules and budgets	Detailed, multi-source cost and schedule estimation; design to cost; incremental development; software reuse; outside reviews.
3. Developing the wrong software functions	Organization analysis; user surveys; prototyping; early users' manual
4. Developing the wrong user interface	Prototyping scenarios; user characterization (functionality, style, workload)
5. Gold plating	Requirements scrubbing; prototyping; cost-benefit analysis; design to cost
6. Continuing stream of requirements changes	High change threshold, information hiding; incremental development (defer changes to later increments)
7. Shortfalls in externally-furnished components	Benchmarking; inspections; reference checking; compatibility analysis
8. Shortfalls in externally-performed tasks	Reference checking; pre-award audits; award-fee contracts; competitive design or prototyping; team-building
9. Real-time performance shortfalls	Simulation; benchmarking; modelling; prototyping; instrumentation; tuning
10. Straining computer science capabilities	Technical analysis; cost-benefit analysis; prototyping; reference checking

Table 1. Top ten software risk items

between NICs and developed countries, for skilled labour in software production. But the introduction and adoption of these new techniques is difficult and expensive, and they are thus adopted bit by bit as project budgets allow.[10]

3.1 Automated software development aids

The conventional "waterfall" software life cycle has been stretched, shrunk and otherwise modified since its inception in the early 1970s. The major stages of software production — specification, design, coding, testing and maintenance — remain, but efforts are under way to alter and automate many aspects of the software life cycle.[11] Lower development and life cycle costs are being realized through better structured and documented software, program support library procedures, diagnostic aids, environmental simulators, test data management systems and applications generators. Such tools and techniques support different phases of the life cycle. Some tools support the early phases in the form of automated diagram drawing, screen painting and error checking. Others give assistance by automating code generation and documentation.

Sizeable applications programs can now be generated using a very small number of user-language directives, which means that developing software with applications generators can be a far more cost-effective pursuit than hiring programmers to develop software one instruction at a time. Applications generators are also valuable for their ability to develop quick prototypes of a desired software capability.

The emergence of applications generators oriented around a database management system and report-generation capability has created an attractive approach for both software productivity gains and for software customization. Some United States and European firms using these tools have improved productivity by factors of two to as much as 20 across various phases of the software life cycle.[12]

The facilities in three different tools — one for complex, real-time computer systems, the other for business applications ("Auto-G", "Excelerator", and the "Cortex Application Factory") now in use are listed below to illustrate the variety of ways in which software automation currently increases the productivity of software of all sizes and types.

Auto-G, produced by a small United Kingdom company, Advanced System Architectures (ASA), is so far the only non-United States system design tool that has been evaluated and supported financially by the United States Strategic Defense Initiative (SDI).[13] The company focuses its operations on high reliability, high security real-time computer systems for tele-communications, aerospace or defence applications.

Auto-G uses a formal graphical notation, G, to enable software engineers to build a system on a workstation in stages, from requirements specification to code generation. When the design is complete, the Auto-G tool set provides a code generator that converts detailed design automatically into a variety of programming languages, including, for example, C and Ada.

The facilities Auto-G provides include:

- Menu-driven, on-screen selection of design symbols;
- On-screen editing;
- Full graphic manipulation;
- Hard copy output to low-cost graphics plotters;
- Automatic checking of logical consistency of design.

Excelerator has four basic facilities for automating systems analysis and design tasks:

- Automatic diagramming tool for drawing structured diagrams, such as data flow diagrams, structure charts, data models and control flow diagrams;
- Screen and report painters for prototyping user interface;

- Integrated dictionary for storing and cross-referencing all systems analysis and design information;
- Automatic checking and reporting the completeness and consistency of structured diagrams.

The Cortex Application Factory is used by systems analysts and programmers to develop and maintain medium-to-large information systems. It is used to lead a developer through the steps of software development and includes features similar to those of the Excelerator, including:

- Screen and report painters for prototyping user interfaces;
- A dictionary for storing documentation about a system;
- Automatic checking for completeness and consistency of program specifications;
- A code generator capable of automatically generating 95 per cent of the program code from program specifications;
- Automatic program documentation generator.

Once completed, the applications are translated into machine code by an optimizing compiler for fast computer run time and greater machine efficiency. Applications developed with the Cortex Application Factory range from such bread-and-butter applications as sales tracking, order entry, accounting, inventory and payroll to more esoteric applications, such as regional price trend tracking for commodities and orders for materials monitoring in bulk fibre manufacturing plants. Applications range in size from a few files, screens and less than 50 data base fields to those with several hundred files and screens and thousands of fields.[14]

Engineered applications generators can produce major benefits in productivity and ease of implementation, use and maintenance *if accompanied with good training and providing unexpected problems do not occur*. Quantitative benefits include reductions in system life cycle costs and rapid development of prototypes to ensure quick turnaround for end user assessment. Among the possible qualitative benefits are the generation of documentation directly from specifications rather than program code, rapid iterative prototyping and greater control over maintenance.

Automated software documentation management techniques are gradually being introduced to software engineering to improve the quality of both software and documentation. It takes an average of three hours to produce a page of software documentation.[15] The cost of one employee's labour-year in the mid-1980s is approximately $US 100,000, so one hour is worth approximately $US 50. Thus, a 500-page software document that may be obsolete by the time it is finished can represent a cost of $US 50,000 to $80,000. Code and documenta-

tion management systems are particularly useful in very large software projects (more than 100,000 lines of code). These systems automate routing and distribution of documentation cross-referencing and status reporting for code and documentation changes.[16]

Some tools now link hardware and software design. For example, an integrated set of tools — C compiler, optimizer, assembler, linker, simulator and debugger — developed by Quantitative Technology Corp. allows engineers to evaluate design iterations of both hardware and software without waiting for a final version of either. A simulation informs the hardware team how design changes will affect software execution speed before a prototype is built, and software engineers can see how code modifications will work on the proposed hardware. Such tools operate on mini- and mainframe computers.[17]

Automated tools alone are not sufficient to ensure a productive software environment. With no coherent methodology, standards and set of controls, automated tools sometimes merely help a software project "spin its wheels more quickly". They can also create an image of modern methods and high technology without substantively increasing productivity.

"Successful" use of automated tools requires a fit between the tool and the environment in which it will be used. The selection of tools can be a complicated and confusing process. Tools must, in some ways, be similar to what a firm already does and knows; they must operate on available hardware and operating systems; and they must be supported and maintained. Firms generally understand and use automated tools only gradually over time.

4. Conclusion

As noted above, one cannot ignore the key technological trends and changes that are now unfolding in software production. Large software firms, especially in western Europe, Japan and the United States, are increasing their degree of software automation and improving their efficiency and performance by better software management practices. Those firms in developing countries and newly industrializing countries that are able to purchase, integrate and exploit software engineering tools will need to be able to provide their customers with working software prototypes and custom software, the latter perhaps at package prices. But this technological capability requires increasing capital, skilled employees (most crucial), access to foreign technology, and an organizational maturity, which does not yet exist in many software firms in both developed countries and in NICs.

Notes

1. See Schware(1989) for some of the implications of basing policy decisions on overly optimistic market projections.
2. Humphrey and Sweet (1987), pp. 5-6
3. Humphrey and Sweet (1987), pp. 23-30.
4. See for example, Humphrey and Sweet (1987) and Kellner and Hansen (1988).

5. Caswell and Grady (1987).
6. Caswell and Grady (1987), p. 25.
7. Caswell and Grady (1987), p.199
8. Caswell and Grady (1987) p. 1.
9. Boehm (1988), pp. 10-11.
10. OTA (1985), p. 76
11. The "US Defense Science Board Task Force on Software" Report (1987), for example, recommends the removal of "any remaining dependence on the assumptions of the "waterfall" model, and to institutionalize rapid prototyping and incremental development". A number of alternative software life cycle models have evolved over several years. However, although each of these alternatives software approaches deals with some of the difficulties of the waterfall approach, especially the fact that the model does not adequately address automatic programming capabilities and "knowledge-based software assistance capabilities", each has its own set of challenges and difficulties to resolve.
12. Business Software Review (1987), pp. 29-30
13. Advanced System Architectures, Ltd.
14. Picardi (1987), p 3. CORTEX Corporation, 138 Technology Drive, Waltham, MA 02154, USA
15. Boehm (1981), p.574.
16. Singleton(1987), p. 54.
17. Young (1988).

References

Advanced System Architectures Ltd., Johnson House, 73-79 Park Street, Camberley, Surrey, GU15 3PE, United Kingdom.

Boehm, B.W., "Software Risk Management: Principles and Practices" (April 1988).

Boehm, B.W., *Software Engineering Economics* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981).

Caswell, D.L., and Grady, R.B., *Software Metrics: Establishing a Company-Wide Program* (Englewood Cliffs, New Jersey: Prentice Hall, Inc. 1987).

Humphrey, W.S., "Characterizing the Software Process: A Maturity Framework" (Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie-Mellon University, June 1987).

Humphrey, W.S., "A method for Assessing the Software Engineering Capability of Contractors" (Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie-Mellon University, June 1987).

Kellner, M.I. and Hansen, G.A., "Software Process Modelling" (Pitttsburgh, Pennsylvania: Software Engineering Institute, Carnegie-Mellon University, May 1988).

OTA, *Information Technology R&D: Critical Trends and Issues* (Washington, D.C.: Office of Technology Assessment, 1985).

Picardi, A., "CORTEX Application Factory: Concepts and Facilities", *Auerbach Information Management Reference* (Boston, Massachusetts: Auerbach Publishers, 1987).

Schware, R., "Trends in the Worldwide Software Industry and Software Engineering: Opportunities and Constraints for Newly Industrializing Countries", Unpublished paper (Washington, D.C.: The World Bank, Room H 3029, 1818 H Street, NW, Washington, D.C. 20433, April 1989).

Singleton, M.E., *Automating Code and Documentation Management* (Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1987).

Young, J.L. "The Software Foundry: Almost Too Good to be True", *Electronics*, (21 January 1988), p 47-51.

Software Technology: Trends

R. Narasimhan*

1. Commodities and Capabilities

In a series of papers and books Amartya Sen, the developmental economist, has been advocating the thesis that economic development is best seen as an expansion of people's 'capabilities'. "This approach focuses on what people can do or can be, and development is seen as a process of emancipation from the enforced necessity to 'live less or be less'". In developing his thesis, Sen draws an essential distinction between capabilities on the one side, and commodities and utilities on the other.[1]

The traditional approach is to measure economic development in terms of availability of goods and services — quantities as well as types. Sen notes that "while goods and services are valuable, they are not valuable in themselves. Their value rests on what they can do for people, or rather, what people can do with these goods and services". Equally, "enhancing utility" interpreted as "enhancing happiness or desire-fulfilment" should not be confused with enhancing capability. In the context of developmental economics, Sen faults "utility-based narrow vision of traditional welfare economics" for its fundamental inadequacy to provide a "basis for evaluating action and policy, in general, and development and structural change, in particular". More generally, the contrast between utility and capability can be seen as the contrast between seeking satisfaction accepting existing constraints as norms, and seeking to change the existing constraints to innovate and explore new possibilities.

The distinction that Sen draws between commodities and capabilities — although worked out in the context of economic development — can be seen to have deep relevance to technology development as well. This is especially true of information technology (IT) where the commoditites/capabilities distinction more or less completely coincides with the hardware/software distinction. In the recent past, all the spectacular breakthroughs in IT have been in hardware (i.e., commodities) and, to some extent, in systems (i.e., utilities). However, we have hardly begun to tap the seemingly infinite potential of information technology to enhance the capabilities of human beings and enrich their life-styles (i.e., quality of life). There are several technical reasons for this. But they can all be summed up by noting that computers have a long way to go before they can become well-adapted companions to human beings. Since software technology (i.e., the software component of information technology) is the essential means for achieving this adaptation, the current

* Professor of Computer Science, Computer Maintenance Cooperation Ltd., Bangalore, India.

trends in software technology can be seen as attempts along several dimensions to bring about human-computer symbiosis in more natural ways. In this chapter we shall discuss some of these attempts, but first we must digress a little to get some feel for what has been happening on the commodities side of information technology.

2. Commodities: Hardware Technology

Here we shall briefly discuss in what sense computer hardware is becoming — if it has not already become — a commodity. We shall not consider the electronics, circuits and fabrication aspects of hardware technology. Some account of these may be found in the papers by Kopetz included in Parts III and V of this volume.

"Between 1980 and 1985, the average end-user price per mips (million instructions per second) declined from about $250,000 to $25,000. .... From 1985 to 1990, the average price per mips fell at roughly the same rate .... from $25,000 to less than $2,500." [2] In 1990, the computer market was divided between the four principal categories roughly as follows: [3]

Mainframe	: 25 per cent
Minicomputer	: 25 per cent
Workstation	: 10 per cent
Personal Computer	: 40 per cent

The market for the first two categories has been steadily declining from the mid-80s, while the market for the other two categories has been steadily on the increase from the early 80s. According to expert predictions: "Except for very large-scale scientific computing ..., all computing will be done with microprocessors; certainly all interesting computers will be micros. So, the microcomputer industry essentially will equal the computer industry".[4] It is also predicted that by the end of the 90s, 1,000 mips capability will be available with desk-top machines.

Notwithstanding the spectacular improvements in the performance of personal computers and workstations, hardware is rapidly becoming a commodity. This view has been most vigorously argued in a recent paper by Rappaport and Halevi.[5] They claim: "Semiconductor value is now a function of specialization. And specialization depends on responsive design and not high-volume low-cost production". Making a similar point, *The Economist* notes that "across the industry, small firms, which have concentrated on doing a few things excellently have flourished at the expense of larger ones, which have taken the conventional wisdom about "solutions" seriously and tried to do a lot of things well". [6]

According to Rappaport and Halevi, "the semiconductor world is on the brink of astounding excess capacity". Semiconductor fabrication technology has become so sophisticated and powerful that "most companies selling high-performance products do not require access to the most advanced manufacturing facilities". What one manufacturer can produce, another

manufacturer can also produce with equal effectiveness. Unless the market is expanded through the creation of new applications — i.e., new ways of using computers and/or applying computers to new tasks — mere improvement in hardware performance will not create additional wealth to the industry as a whole.

Taking laptops as a specific example, Rappaport and Halevi argue that: "laptops themselves — even though they represent a major innovation in raw computing power — do not create a new computing paradigm. Without new applications they will merely replace rather than expand the computer hardware market".

If making good computers is not enough to bring success, what additionally needs to be done? "Adding utility" is the answer to this question according to Rappaport and Halevi. To add new utility one has to look-ahead, enlarge one's vision, and identify possibilities for value-addition through new concepts and by devising new technologies based on these. Some examples suggested by Rappaport and Halevi are: "pen-based operating systems, natural language interfaces, multimedia data storage and recall, and so on". Practically all such extensions to information technology constitute challenges in software innovation rather than in new hardware development. This leads us then to look at software technology as a resource for endowing the commodity side of information technology with capabilities. We shall discuss some recent developments in software technology from this perspective.

3. Capabilities: The Software Environment

It is the software environment that converts the commodity-orientation of information technology to the capability-orientation. To understand how this comes about, it is not sufficient to look upon a computer as [hardware + software]. It is essential to analyze 'software' into its several functionalities, which together constitute the software environment for computing. Figure 1 is an illustration of such an articulation.

In terms of this articulation, the trends in software technology may be briefly summarized as follows. One major thrust is in simplifying the human-machine interface and making it more natural for use by human beings. Another thrust is to move computing away from the linear symbolic domain to a spatial visual domain. There is much investment in devising sophisticated visualization techniques. Simultaneously, there is a move to enhance interaction through the incorporation of motion pictures, sound, and body movements. Coming to grips with "virtual reality" is an increasing preoccupation in spatially-oriented computing.

To support the above innovations, enabling developments in database technology are essential. It is important to move away from exclusive preoccupation with textual databases and to begin to cope with databases of other kinds as illustrated in Figure 1. The ability to traverse smoothly a mixture of such databases is a prerequisite to functioning realistically in a multimedia environment. This capability would seem to be closely related to having some understanding — in a conceptually manipulable sense — of the domains which these different data types model.

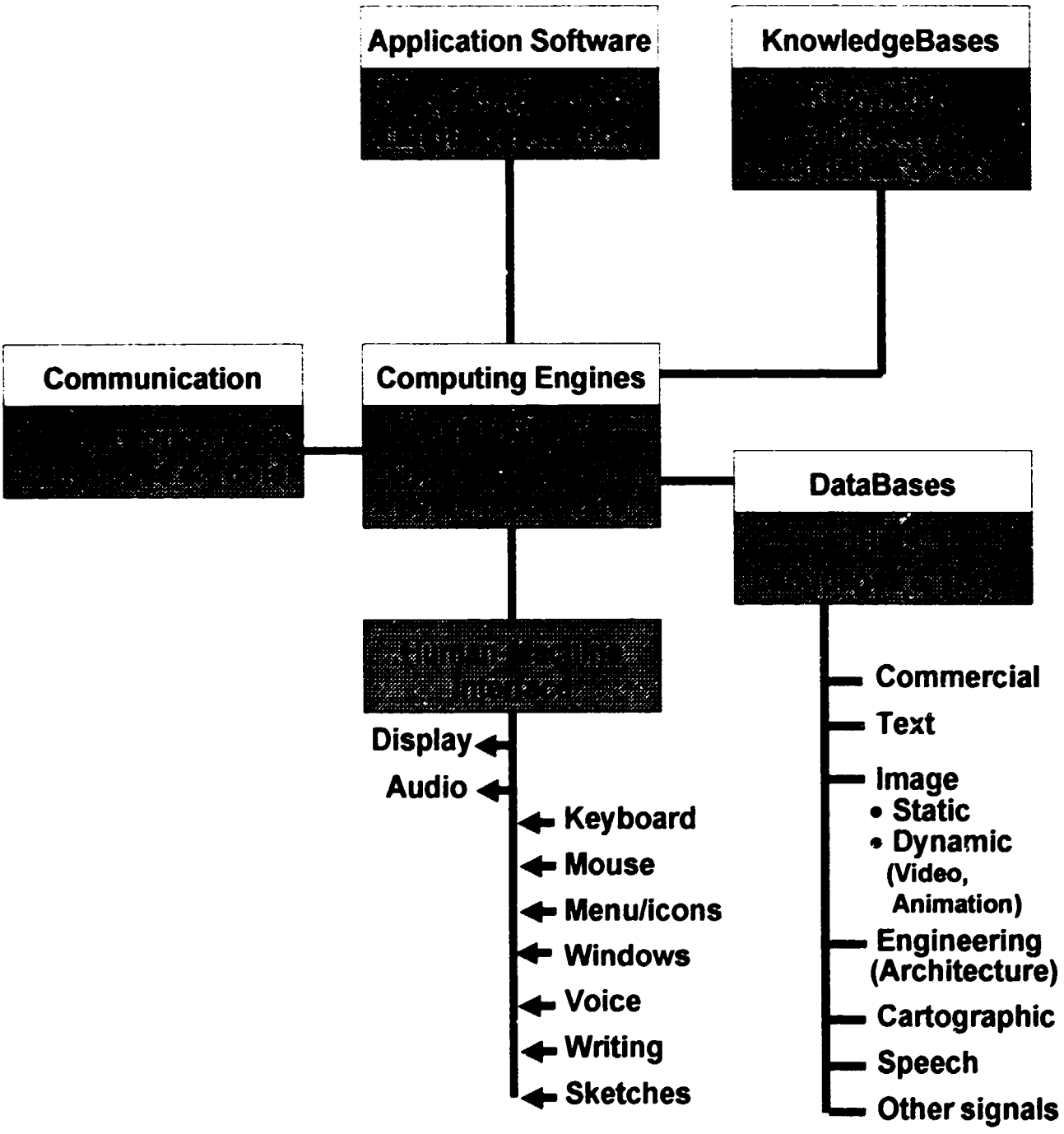


Figure 1. Components of a Computing Environment

Even the most user-friendly computer is at present too difficult to use for most people who are not computer specialists. As has been remarked: "The most profound technologies are those that disappear".[7] They become a pervasive part of one's environment and the usage of that technology becomes totally interiorized. For literate communities, 'writing' is such a technology. In this sense computers have a long way to go before they become "an integral, invisible part of people's lives". Mass markets for computers — especially in developing countries — cannot be expected to develop until computing can be totally interiorized in this sense.

Interacting with computers, at present, requires too detailed a specification to be provided by the interacting human being of what he or she wants the computer to do. In our normal mode of interacting with other human beings, we leave many things unsaid. Shared contexts and shared presuppositions help to fill in the gaps left unspecified. The ability to do this is the basis of commonsense. Computers currently lack commonsense. How to endow them with some semblance of commonsense is a major research issue. Knowledge, intelligence, capability to conjecture, collect evidence, reason, judgement, and so on, are various facets of common sense. Knowledge-based computing is an effort to come to grips with these capabilities, which we take for granted tacitly while interacting with other human beings. Reasoning, which is the foundation of intelligence, would seem to be an essential capability that computers must be endowed with. Artificial intelligence research and knowledge-based computing studies are centrally concerned with this issue. Preliminary successes in these efforts are already being spun-off into software technology practices.

The software generation process is becoming more and more automated and tool-intensive: (see the papers by Turksi and Schware in this part). With the coding process more or less fully automated, the focus of systematization will move to the specification process. This is already happening through the invention of specialized specification languages. But for major breakthroughs in this area, we have to understand the application areas more analytically. A natural generic taxonomy of application areas might make it possible to develop schematized architectures well-tuned to specific application classes. Such schematizations should enable rapid prototyping of application software and vastly improve software productivity.

Networking of computers has gone through rapid developments during the past two decades. The technology of distributed computing is beginning to be taken for granted as a normal mode of accessing computers and engaging in cooperative software development activity. The client-server paradigm separates the computing related to the user-interface at one end from the back-end activities associated with centralized information stores. These latter have been 'down-sized' to use superminis. The client-server technique has vastly reduced the amount of network load involved in performing a transaction and has vastly improved the man-machine interface use in on-line transactions. The end result has been the effective exploitation of the rapidly falling costs of personal workstations and local area network (LAN) interfaces.

The availability of high bandwidth channels at affordable prices is making it possible to use computer networks for multimedia applications. Voice mail is soon bound to become as familiar as e-mail on networks. Distributed computing and real-time computing are also providing an impetus for advanced developments in the architecture of operating systems.

In the following sections we shall consider in somewhat greater detail, developments in software technology directly concerned with augmenting the capabilities of computers to interact with human users in more natural and friendly ways.

4. User Interface

User interface is the channel that links a user with a computer when the computer is being used in an interactive mode. Typically, interacting with a computer a step at a time requires the user to specify a desired action to be carried out (by the computer) on designated objects. Simple examples are: *delete* a file (identified by its name); *create* a (new) file assigning a given name to it; *copy* one designated file into another designated file; *execute* a program (identified by its name) with specified arguments (identified by their names); and so on.

At each step, the user typically has to know the action options available to him at that stage of the interaction, and for each action that he selects, the arguments (i.e., objects) that need to be specified. The problem of making the interaction interface user-friendly consists in providing the right kind of assistance to the user at each interaction step. Productivity improvements, job satisfaction, avoiding fatigue, increasing motivation and so forth, are all ultimately determined by how easily and *meaningfully* a user can interact with a computer.

In the last ten years or so, the single factor that has qualitatively improved the user interface, making it enormously user-friendly, is the shift away from the textual mode of interaction to the graphical mode of interaction. This shift in modality can be characterized as moving away from "telling" to "pointing" — from *telling* what is the desired action to *pointing* what is the desired action. Instead of expecting the user to carry in his head a whole mass of detailed information about the current state of interaction, the options available and the specific command to be keyed in to invoke a particular option, all these details are *displayed* to the user on his terminal through imaginative use of *icons* and *menus*. A hand-held device — typically a *mouse* — is used to point and click, thus selecting and invoking an action, or choosing an argument (object) for an action. The display is continuously kept updated to reflect the interaction path the user has traversed so far.

This form of a graphical user interface (GUI) was pioneered by scientists at Xerox PARC in the 70s resulting in the Xerox Star computer. These ideas were subsequently picked up and made into a commercial success by Apple Computer for use with its Lisa and, later, Macintosh models. With the proliferation of IBM-compatible PCs, the GUI has been extended to them by several software vendors, some of the more widely-used being *Windows* (from Microsoft), *GEM* (from Digital Research) and IBM's *Presentation Man-*

ager to go with OS/2. Similarly, the *X Window System*, first developed at MIT and now actively supported by the MIT X Consortium consisting of almost all graphics workstation manufacturers, provides a portable standard for developing GUI applications on graphics workstations.

Recent research in the area of GUI is focusing on a variety of issues. Much effort is going into the systematization of the theory and practice of the design of user interface management systems.[8] The original desktop metaphor (which resulted in the Macintosh and Windows type of GUIs) is being extended, again by scientists in Xerox PARC, to cope with interaction environments that cannot be confined to a single desktop. "Instead of one big virtual desktop, the PARC team is trying out a new metaphor for organizing the world inside the computer: rooms. Each room contains the tools and data needed for a different sort of task. To switch tasks, move from one room to the next. In the *Information Visualizer*, moving from room to room looks like moving down a set of corridors".[9] Yet another attempt being tried at PARC is to organize multiple screen data in terms of 3-D spatial architectures; for example, walls of a room or set of rooms displayed in perspective projection to the user.

So far we have been concentrating on the GUI aspects relating to display, presentation of information relevant to each step in an ongoing interaction, and choice of actions to move the interaction step by step. Equally important in determining the friendliness of a user interface are a variety of ergonomic and human factors. There is a large literature describing and analyzing these factors and discussing how to make the right design choices in structuring user-friendly interfaces. [10]

Aside from alpha-numeric keyboards and indirect pointing devices such as a mouse, joystick, etc., direct pointing devices such as a light pen, or touch screen, could be more appropriate in particular interactive applications. Voice input is bound to become a valuable adjunct to a friendly user interface. But commercial realizations of voice inputs are, at present, seriously restricted in their scope. Computational problems relating to speech recognition and speech understanding are still mostly unsolved and/or ill-understood. User interfaces incorporating unrestricted voice input are unlikely to be available for general use for quite some years.

Like voice inputs, *writing* is a more natural interaction mode than typing for most literate human beings. Taking note of this, pen-based computers are beginning to be commercially exploited. However, unrestricted handwriting recognition remains an unsolved problem. Hence, open-ended pen-based computing is not likely to become a commercially viable technology in the near future. However, pen-based computing has been attracting the attention of several hardware and software manufacturers. Pen-based operating systems have been developed by at least two software houses — Microsoft, and Go-Corporation of California. *Pen-Window* of the former uses a pen-like stylus for pointing and clicking instead of a mouse. Go's *Pen point* uses a notebook metaphor instead of a desktop one. A notebook has information gathered into pages, with index markers and a contents page.

"The user operates the software by touching the pen to a page number or an index marker displayed on the computer screen ..... *Penpoint* also recognizes printed letters and numerals, upper and lower cases, and some common editing marks".[11] Limited textual information can be entered by printing characters with the pen — a necessarily slow process and, hence, not intended for long texts.

A natural extension to writing with a pen is to draw sketches and other notational symbols on a tablet with a pen and have these read and interpreted by the computer. These modes of interaction are, in a deep sense, more natural to human beings than typing or the use of string languages for formulating commands. Interfaces incorporating such modalities are bound to be more user-friendly. However, to successfully cope with the recognition and interpretation problems intrinsic to such interaction modalities, the market may have to wait for a wider and more general availability of intelligent and knowledge-based computing.

Endowing computers with knowledge and intelligence is, in fact, the bottom line in making computers user-friendly in a human-like sense. We have a long way to go before we can successfully incorporate common sense and reasoning capability in computers. However, the results of three decades of research in artificial intelligence (AI) are already making it possible to design user interfaces with some knowledge-based skills. "Aldus, which makes desktop publishing (DTP) software, is said to be trying to incorporate a small expert system in its bestselling *Pagemaker* software to help advise people on how to make better-looking documents".[12] Expert system technology, in general, would seem to have the potential to understand the "intention" of the user and base advice on this knowledge.

A computer system capable of providing intelligent advice in this sense to a user must have two kinds of knowledge: first about the application domain the user is grappling with (for instance, DTP)in the Aldus case above), and second about its own capabilities, i.e., the services it can provide. But it is not sufficient to make available this knowledge in the form of "HELP" files. The system must be able to *reason* about its own actions and those of the user; infer the intentions of the user (that is, what he is seeking to accomplish); and help him along in the right direction. Clearly, humanizing the user interface through the availability of voice, writing, sketching, pointing, use of natural language, and so on, are only means to this ultimate end.

5. Multimedia

User-friendly interactions with computers in the future will call for the ability to cope simultaneously with information in a variety of media. That is, not just with texts and graphic symbols presented on the computer screen (as we have been discussing so far), but with speech (natural, as well as computer synthesized), video segments in colour (both still images and moving pictures), computer animated sequences, sound, geographical and spatial information such as maps, architectural drawings and so on.

For multimedia applications to be accessible to casual computer users, appropriate hardware platforms would have to be devised and these must be available at affordable prices. Specifications for PCs to support multimedia applications have already been defined and PCs satisfying these specifications are beginning to be available. "Proponents claim that by the late 1990s, such multimedia PCs will have four "Gs": gigabyte of main memory, a gigabyte (at least) of secondary storage, giga operations per second, and gigabit-per-second data transfer rates".[13] Will the multimedia technology make its first major impact through the consumer electronics market or the computer market? This is a hotly debated question in the industry. Current indications are that the consumer electronics market, perhaps, has an edge over the computer market for wider acceptance of this new technology. The reason for this is clear. With the current state of the multimedia technology, it is easier to create products focused on entertainment, and to a limited extent, education, than it is to create applications for specialized professional end-users.

The advertisement industry and the engineering industry can derive significant benefits from the new technology. It has been suggested that the expected drop in the prices of 3D graphics workstations would open up exciting new opportunities "... to combine digital video and 3D graphics in advertising". "Architects and designers will be able to use multimedia systems to call up typical designs from a multimedia repository, customize the design to meet a job on hand, visualize the design in 3D, make a video clip, and send it to the customer". [14] Multimedia technology offers great potentials for exploitation in education and training — especially, training in diagnosis and repair, training in assembly and shop-floor operations, training of paramedical personnel, and so on.

However, before multimedia technology can be effectively deployed to support these kinds of applications, efficient application development methodologies must be devised. In attempting to tackle this problem, one has to come to grips with issues that are not only concerned with computing or computer science. Creating a multimedia product is akin to creating a movie — a T.V. film, for example. Scripting, designing, typography, visualizing, editing, sound and music, all play vital roles in realizing an effective end product. Teams of experts in each of these areas have to work together to create multimedia products. Managing such teams would call for project managers with a combination of skills: (1) in multimedia technology; (2) in communication; and (3) in visualization, art and aesthetics. A good multimedia application methodology has to address issues relating to all these domains.

Handling multimedia technology effectively calls for new developments in database technologies. As has been noted: "new conceptual issues are encountered as attempts are made to apply database technology to types of data not previously stored in databases, such as speech, signals of various types, texts, geometric data (e.g., engineering design and cartographic data), imagery, AI knowledge representations. Such data types frequently involve

highly specialized data structures with very large storage requirements and with processing programs that often access data in patterns for which conventional database techniques were not designed". [15]

Confining our attention for the moment to digital audio and video components of multimedia, the first problem to face is the enormous volume of storage needed to represent them to ensure high quality reproduction. A variety of compression techniques have been worked out and new ones are continuing to be invented. Compression techniques can operate at the signal level. These are the ones, which at present are being widely used commercially, and some of these have already been incorporated into silicon chips. But as Fox points out: "Ultimately, images and video will have to be analyzed and stored in high level, storage-efficient representations that identify and characterize objects, relationships, distances and movement."[16] Such techniques lead to model-level compression supporting truly scalable presentations of displays varying from palm-size to wall-size. Clearly, before model-based compression techniques can be devised, we have to acquire a deeper conceptual level understanding of stationary as well as moving images. in order to be able to formulate adequate models of them. We are very far from achieving this at present.

Creating suitably indexed repositories of digital audio and video objects (sequences or files), and structuring these in the form of databases of interlinked entities for rapid retrieval and editing, is a prerequisite to large-scale multimedia applications development. As Fox points out again: "While many people know how to file text documents, select a graphic object while drawing, or choose slides for a presentation, most people have had no experience editing audio or video files. Nevertheless, tens or hundreds of hours of raw footage are often edited to produce a single hour of video presentation." [17]

Editing text files under computer control interactively has become a sophisticated technology. Here, one is dealing with linear strings of symbols, moving them around, deleting, interleaving, permuting, and so on. In the case of audio, video, and spatial structures such as maps, architectural drawings, etc., editing involves operations in the temporal and spatial domains. It is unclear whether one can arrive at universal (i.e., domain independent) editing primitives in these cases. The entity-relationship models of standard database techniques clearly are inadequate. Editing in the temporal and spatial domains call for complex transformations to be performed on objects (temporal/spatial entities) and not just their selection and rearrangement as in the case of strings. Editing commands have, of course, been developed for use in computer graphics. But conceptually new approaches may have to be devised to exploit the full potentials of multimedia.

6. The Future

Looking upon computers as tools that extend human cognitive and communication capabilities, we have discussed so far developments along two dimensions:

1. Making computers more user-friendly, and

2. Enriching the semantic and representational aspects of the information environment provided by computers for interactive usage.

Multimedia and highly dynamic interaction together enable qualitatively new ways of visualizing and exploring aspects of physical phenomena hitherto inaccessible to the human senses. The computer acts as a window providing a view of "virtual reality" made accessible through simulation for realistic interaction. Many of the desiderata we identified in the last two sections, one can be sure, will be realized as actualities in the not too distant future.

But what can we say about the distant future and beyond ? Alan Kay predicts that the next stage of evolution will see computers transforming themselves from being mere tools to becoming agents.[18] Tools are passive and are there to be manipulated by humans (i.e. agents) to reach desired objectives. The computer as an agent is a robot. It is no longer a passive entity but is capable of initiating actions on its own. Minimally a robot is programmed to carry out a pre-determined sequence of actions under environmental control — including explicit commands externally given. Kopetz's contribution included in section V of this volume argues the desirability of incorporating intelligence in control and measuring equipment, and other household and engineering gadgets. Specially programmed microprocessors act as agents in these cases.

Weiser extends this scenario to its ultimate possibility while describing the explorations he and his colleagues are conducting at Xerox PARC[19]. He calls the scenario "ubiquitous computing" or "embodied virtuality". Consider first the case where a larger and larger number of artifacts in one's normal living environment (either in the house or in the office) have agents (i e. programmed microprocessors) embedded in them. Next, assume that all these agents are networked through acoustic, radio, infrared, or whatever means. Finally, consider that interactive computers of various sizes (notepads, terminals, black boards, and so on) are available scattered around for ready access, and that these are part of the ubiquitous network earlier described. Assume that these computers are also networked to geographically distributed information sources, including other computers. What kinds of life styles would such ubiquitous computing environments make possible ? Weiser describes an illustrative example in his paper.

While Weiser's ultimate scenario might seem too much like science fiction, restricted networking of distributed agents is, clearly, operationally viable and within technological reach in the foreseeable future. Minimally we can expect individual PC's, workstations, or whatever, to function at the same time as a networked computer, a radio, a T.V., a telephone, a hi-fi sound system, VCR, a fax machine, a telex machine, a clock. Add-on cards to PC's already make available many of these functionalities. Next, add to these other multimedia functionalities and access to packaged information resources such as books, encyclopedias, databases, and so on. A PC or a workstation becomes, in these circumstances, a very versatile and informationally rich work (or entertainment) environment. Adaptations of

such environments to cater to particular professional groups (engineering, medical, legal, advertisement, newspaper, etc.) are the kinds of capability developments most likely to become commercial realities in the next decade or so.

Acknowledgment

I would like to thank the National Centre for Software Technology, Bombay, and particularly Dr. S. Ramani, Dr. S. P. Mudur, Mr. P. Sadanandan and Mr. R. Chandrasekar, for extensive assistance in the preparation of this chapter.

NOTES

1. The principal reference is Amartya Sen (1983). The quotations here are from Sen's paper "Goods and People" included in the collection Amartya Sen (1984).
2. Rappaport and Halevi (1991), p. 70.
3. *The Economist*, 2 November 1991, p. 68.
4. This and the following prediction are from the interviews with experts compiled by *BYTE* in its September 1990 issue; pp.226, 234.
5. Rappaport and Halevi (1991).
6. *The Economist*, 2 November 1991, p.67.
7. Weiser (1991), p.66.
8. For an excellent review, see Ero and van Liere (1988).
9. *The Economist*, 29 June 1991, p. 80-81. This report is a readable summary of recent work on GUI at Xerox PARC.
10. For a good review and extensive references to literature, see Ero and van Liere (1988).
11. *New York Times* news story reprinted in *Deccan Herald*, 30 May 1991.
12. *The Economist*, 14 March 1992; p.7 of an excellent 20 page survey article on the current status of AI.

13,14 Narasimhalu and Christodoulakis (1991). These quotations are from the introduction to the special issue of *IEEE Computer* (October 1991) on "Multimedia Information Systems", guest-edited by them.

15. Mylopoulous and Brodie (1989). This quotation is from their introduction.
16. Fox (1991), p.12. This paper by Fox included in the special issue referred to above in 13/14 is an exceptionally informative review of the status of interactive digital multimedia systems.
17. Fox (1991), p.11.
18. Kay (1990), p.241.

19. Weiser (1991), p.66-75.

References

1. J. Ero and R. van Liere (1988): "User Interface Management Systems"; in M.M. de Rinter (Ed.): *Advances in Computer Graphics III*, Springer-Verlag, Heidelberg; pp. 99-131.
2. E. A. Fox (1991): "Advances in Interactive Digital Multimedia Systems", *IEEE Computer*, October, pp. 9-21.
3. A. Kay (1990): "On the Next Revolution"; *BYTE*, September, p. 241
4. J. Mylopoulous and M. L. Brodie (Eds.) (1989): *Readings in AI and Databases*, Morgan Kaufman, California.
5. A. D. Narasimhalu and S. Christodoulakis (1991): "Multimedia Information Systems: The Unfolding of a Reality", Guest Editors' Introduction, *IEEE Computer*, October.
6. A. S. Rappaport and S. Halevi (1991): "The Computerless Computer Company", *Harvard Business Review*, July-August, pp. 69-90.
7. Amartya Sen (1983): *Commodities and Capabilities*; North Holland, Amsterdam.
8. Amartya Sen (1991): *Resources, Values and Development*; Basil Blackwell, Oxford.
9. M. Weiser (1991): "The Computer for the 21st Century"; *Scientific American*, September, pp. 66-75.

III

The Software Market

The Software Market: Emerging Trends *

Hermann Kopetz **

1. Introduction

In the last few years, software production has emerged as a major industry of substantial economic significance. It is estimated that over half of the worldwide information processing market in the order of more than US$ 500 billion is, in one way or another, related to software production. This market is still growing at a rate of about 10 per cent per year. Millions of software professionals, system analysts, programmers, managers, etc., are engaged in software production worldwide. It is our opinion that every industrialized or developing country in the world has to face the impact of the software industry. Software can be seen as an industrial product, which is imported to serve the local needs or which is locally produced and exported to the world market. In a maturing economy, which is integrated into the economic systems of the world, software should be seen as both an import and an export product.

Most of the software produced does not appear on the open market in the form of prepackaged software. A formidable amount of software is produced by the computer companies and sold together with their hardware as a computer systems product. A lot of software is integrated in other tangible products to increase their functionality and value. Many of the latest consumer products and industrial products contain integrated microprocessors or computer systems with significant amounts of application software hidden behind a user-friendly man machine interface. Software is also needed for the computer integrated manufacturing (CIM) systems. Computer-based production planning and control systems are installed in most manufacturing plants. It follows that a state-of-the-art software capability is not only needed for the production of software *per se*, but also in the production of any kind of industrial product.

Software is a know-how intensive industry. Highly qualified, well trained and fully motivated personnel form the basis of any software industry. Because productivity ranges are much higher than salary ranges, it is wise to attract the best people to software production by offering extraordinary working conditions.

* Excerpted from UNIDO IPCT.144(SPEC.), Nov. 1991

** Professor of Computer Science, Technical University of Vienna, Austria.

2. The Operational Environment

An end user is interested in the delivery of some specified computational service of an integrated computer system. Only the proper combination of computer software and hardware can provide such a service. From this point of view, software in itself is not a complete product — it requires an operational hardware environment to produce the intended effect. In this section we review the expected trends in the field of computer hardware and operating systems to sketch the operational hardware environment for future software.

2.1 Hardware trends

In the next ten years the technological advances in the field of computer hardware will continue to produce more powerful microelectronic chips at the same rate as we have seen in the recent past. These advances will affect both the functional capability and performance of computer hardware.

By now it is fairly clear that the next generation of memory chips, the 64 Mbyte chip, will be introduced in the market around 1995. Before the end of this decade the following generation, the 256 Mbyte chip, should be available. Similar advances can be expected in the capabilities of the Central Processing Units (CPU) and storage media. The interconnection of computers will be realized by high speed networks in the gigabit range.

A short glimpse backwards should help us to put these extraordinary developments in the proper perspective. When the first personal computers were marketed at the beginning of the eighties, a memory size of 64 kbyte was common. Five years later the typical memory has increased to about 512 kbytes, while today, another five years later, another ten-fold increase to 5 Mbyte can be observed. If this trend continues — and there are convincing indications it will — a typical personal computer will have a memory of more than 100 Mbytes before the end of this decade.

More importantly, the expected advances in the field of VLSI integration will allow the integration of more than 10 million circuit elements on a single chip before the end of this decade. This level of integration makes it possible to manufacture complex information processing systems on a single chip. Powerful single chip microcontrollers with onboard RAM, ROM and process I/O will be available for all kinds of integrated control functions.

Since the design of highly integrated VLSI chips is extremely costly and the marginal manufacturing costs are relatively low, only a few "families" of general purpose processors will dominate the worldwide computer market. This has important implications for the software industry. Only a small number of standardized operating systems will be available.

Over the next decade, the cost of producing computer hardware of a given functionality will decline at a similar speed to that of the past decade. An equivalent reduction of the software

costs cannot be expected. Therefore, on a system basis the strategic importance of software will increase, i.e. the major fraction of the value of an integrated computer system will be in its software.

2.2 Operating systems

The standardization of the operating systems is well under way. Proprietary operating systems from the major computer companies are giving way to standardized operating systems, such as UNIX and MS/DOS. According to a recent United States statistic concerning the personal computer market, [Bul91] the following operating systems were dominant by the end of 1990:

MS DOS	60760
MS Windows	8860
Apple Macintosh	5051
UNIX	1500
OS/2	700

Major PC Operating Systems shipped through year-end 1990

(in thousands of units) [Bul91]

From these numbers it is evident that at the moment the MS/DOS market is the most interesting market for producers of commercial software packages. An established operating system base of more than 50 million installations offers tremendous opportunities for a novel software package of mass appeal.

In the field of real-time software and microcontroller software, no standardization tendencies similar to the PC market are observable at the moment. Although the United States Department of Defense launched a standard programming language and environment, ADA, for embedded computer applications some years ago, this language has not yet found widespread acceptance outside the military community. Many embedded computer applications are based on small proprietary real-time operating systems of focused functionality.

The future trends in the operating system field will be determined by a number of factors. The expected hardware environment, as described above, will open completely new markets, such as multimedia applications integrating text, sound and video. It is not sure whether the present market leader in operating systems, MS/DOS can evolve to handle these new applications in an optimal way.

On the other side, new strategic alliances are being formed in the computer and electronics industry worldwide. The collaboration between IBM and Apple computers can have a

considerable effect on the future of the MACINTOSH operating system. The agreement between Digital Equipment Corporation and Philips can have an impact on the multimedia market, which is also at the focus of the major Japanese electronics companies.

3. Software Products And Services: Their Nature And Market

For the purpose of our analysis of the economic opportunities of developing countries in the software world, we will partition the software activities into the following five segments:

1. Prepackaged software
2. Key element software
3. Intelligent software
4. Contract programming
5. Software maintenance

Although these segments are partially overlapping, they each have a clear identity, as will be seen in the following discussion.

3.1 Prepackaged software

Prepackaged software is defined as that segment of the software industry which is dedicated to the production, marketing and maintenance of software packages, such as a word processing program.

Although there are many thousands of small software companies trying to establish themselves in this very competitive market, only a few companies succeed. The best-selling application packages are all based on the MS/DOS operating system. According to a US statistics [Bul91], the packages that make the list of the top 10 prepackaged software products by the end of 1990 are as given alongside. Note that none

	Software	Company	Units sold
1	Windows	Microsoft	8,860,000
2	Lotus 1-2-3	Lotus Develp.	8,000,000
3	WordPerfect	WordPerfect	5,500,000
4	WordStar	WordStar Int.	4,200,000
5	dBase	Ashton-Tate	3,315,000
6	PCPaintbrush	ZSoft	2,050,000
7	Multiplan	Microsoft	1,800,000
8	SuperCalc	Computer Ass.Int.	1,700,000
9	Norton Utilities	Symantec	1,500,000
10	Deskmate	Tandy	1,500,000

of the packages based on the second most successful operating system, the MACINTOSH, made it to a place in the top 10 best-sellers.

These numbers, impressive as they are, should not lead to the false conclusion that prepackaged software is the only major market for the output of the software industry. Although it is the most visible market, it accounts for less than 10 per cent of the total output of the software industry. The world's leading software company, MICROSOFT, which produced the leading PC operating system MS/DOS and has two products in the 10 best-sellers list above, recorded sales of about US $ 1200 million last year, less than 2 per cent of the sales volume of the leading integrated computer company, IBM [Bus91].

Successful products in the "prepackaged software" market address the following issues with great care:

(1) Genuine user need: A lasting success of any product depends on the relevance and quality of service it can provide to its users. The focus on the true needs of the prospective users is thus the foremost requirement of a software product.

(2) Ease of use: A product aimed at a mass market should be usable by an average user without extensive training. "Ease-of-use" is a complex software characteristic which not only depends on the product *per se*, but also on the background and experience of the prospective use. Many new software companies fail because they underestimate the effort required to make a program easy-to-use.

(3) Documentation: A mass software product has to be accompanied by a flawless documentation that is so organized to answer all conceivable questions from its users quickly. Preparing such a documentation is a major effort, which is often underestimated.

(4) Quality: A serious error in a mass software product can nullify an expensive marketing campaign and cause excessive support costs.

(5) Support: Vendors of successful software packages provide extensive after sales support to their customers. It is common that the average customer will require more than one telephone based assistance directly from the supplier, since many distributors are not in the position to answer in-depth questions from customers.

There are many instances of successful software products in small specialized markets. These products are based on comprehensive application know-how in a specific area and

provide excellent service for a select customer base. These products are not necessarily linked with all the key attributes of "mass market" software, e.g. a delighted customer might be willing to trade some "ease-of-use" for a profound assistance in his key business needs.

3.2 Key element software

Key element software is defined as that segment of the software industry where the software contributes the key element to a complex industrial product, e.g., the software for a telephone switching system or an operating system for a new computer. A typical example is where an existing industrial organization with an established home market tries to improve or enhance its existing product by new functionality based on software.

The key element software market is dominated by major industrial companies both inside and outside the computer field. The major computer companies have to develop or adapt the system-software and networking-software to their hardware architecture, resulting in an immense software effort.

Our attention will focus on companies outside the established computer field. Many of these industrial companies replace their conventional control systems by software based subsystems and add improved functionality to increase the competitiveness of their products in their respective markets. The cost of the software subsystem can range from a small proportion (a few per cent) to a substantial part of the final product price. Characteristic for the key element software is the rapidly rising share of the software subsystem. Let us consider the automobile. A few years ago there was hardly any software based subsystem in an automobile. According to industry estimates, by the end of this decade 10 to 15 per cent of the cost of an automobile will be in the electronics subsystem, a considerable part thereof in the software. A similar trend can be observed in avionics control systems, telephone switching systems, etc.

The following attributes characterize the key element software:

Dependability: Since many of the targeted applications, e.g., an avionics system, can potentially fail in catastrophic failure modes, the software must be highly dependable and support fault-tolerant operation. Other applications, e.g., telephone switching, require extreme levels of availability (e.g. less than two hours of outage in 20 years of operation).

Real-time response: Most of the software in this market segment is concerned with real-time applications, i.e., it must be guaranteed that a result will be produced by the computer within the specified time interval.

Complexity: Many of the applications in this market segment are inherently large and complex. It therefore requires a substantial invest-

ment in a sizable software development organization and quality assurance program to meet the demanding requirements of the applications.

3.3 Intelligent product software

"Intelligent products" are products which integrate a mechanical subsystem with a computer controlled subsystem into a compact functional unit thereby fulfilling a specific user need, e.g. an automatic scale with an integrated microcomputer to perform the calibration, weighing and recording functions. A substantial portion of the cost in the intelligent product is in the application software development. The application software forms an integral part of the intelligent product and is normally stored in a Read Only Memory, i.e., it is integrated in the hardware and cannot be changed easily.

In a recent contribution to the UNIDO *Microelectronics Monitor*, January 1990 ([Kop90]), the characteristics and design challenges in the production of intelligent products in developing countries was analyzed.*

Although intelligent product software is related to the key element software, there are substantial differences, which justify its separate classification. The following attributes characterize the intelligent product software:

Dependability: The reliability of an "intelligent product" may not be compromised by errors in the software. Since the software is integrated in the intelligent product it is not possible to correct software errors in the field, i.e., in case of a software error the whole product may have to be discarded.

Real-time response: The software in most intelligent products must respond in real-time.

Complexity: The inherent software complexity of intelligent product software is normally much lower than the complexity of key element software.

Optimal resource utilization: Intelligent products are sold in a mass market in large quantities. Therefore, it is necessary to be aware of the resource requirements of the software.

Because of the lower complexity and stand-alone utility, intelligent products are well-suited for development by innovative small companies.

* This contribution is reproduced in Part V of this volume.

3.4 Contract programming

Contract programming is concerned with the design and implementation of software relative to given functional specifications. In a typical scenario, a client, i.e. an industrial company, specifies a software package and formulates a set of acceptance tests. A software service organization, e.g., a small software house, implements this software package and delivers it to the client. Normally the software house will also offer a maintenance contract for the delivered software.

Contract programming can be an interesting activity for developing countries. It requires little capital investment, other than excellent training facilities and access to an international electronic network, e.g. Internet. There is a significant potential for cooperation between industrialized and developing countries in the field of contract programming. A major industrial company can start with small contract programming projects and can gradually build up a partnership with an organization in a developing country. There are some developing countries, notably India, which are active in the field of contract programming.

A small software house in a developing country, connected to one of the major international computer networks, e.g., Internet, can receive the specification and deliver products electronically. Also the interactions between the client and the software house can be executed via e-mail so that an interactive dialogue is possible.

3.5 Software maintenance

Software maintenance encompasses the elimination of software errors, the adaption of existing software to a new hardware platform, and the enhancement of existing software. It is performed either by EDP departments within organizations or by independent software houses closely linked to an organization . Software maintenance is the largest software activity. Its size is related to the general level of industrialization and automation within a country. Although its volume is still increasing in absolute terms, it is decreasing in relative size. Many proprietary software solutions within companies are replaced by prepackaged softwares. As a consequence, the need for software maintenance is reduced. Software maintenance requires a good understanding of and a smooth cooperation between the organization utilizing the software and the organization providing the service. The maintenance activity is normally not open for outside competition.

4. Software Distribution

After a new software product has been developed successfully it has to be distributed to an interested clientele. We will distinguish between three different styles of software distribution: commercial software, integrated software and free software.

Since software is an intellectual product and can be easily copied, the intellectual property laws have been expanded to cover software and thus protect the software developer. Still, a

sizable fraction of prepackaged software is copied illegally. The present situation concerning the intellectual property rights of the software owner is still a subject of heated debate. On the one side it is felt that the protection of the software rights, e.g., of the "look and feel" of a man-machine interface, is not sufficient to protect the ideas of the original inventor. On the other side it is argued that too often copyrights and patents are granted to programs that are obvious. As a consequence programmers must spend more and more of their time on finding ways around existing patents. It is feared that in the future it will be difficult to write useful software because most basic ideas are protected and every new program is likely to infringe patents.

There is some logic in the radical idea, expressed by a minority group, to reduce the scope of the software protection; software companies should make their money by servicing the software products they create. Such a measure would reduce the increasing number of costly litigations and eliminate the legal barriers on creativity in the software field.

4.1 Commercial software

The standard way to market prepackaged software is via wholesale and retail distributors, such as a computer shop or a mail-order company selling via a software catalogue. Since there is only a limited amount of shelfspace available in a computer shop, it can be very difficult to convince a computer shop owner to display a new software product if at has not been widely advertised. Expensive marketing campaigns are necessary to bring the new product to the attention of prospective buyers. In many cases, the cost of distribution is significantly higher than the cost of software development. There are only a few small software companies who can afford the prohibitive cost of an effective product introduction.

Once a new software product is established in the market it is enhanced and improved, based on the response of the customers. A new version is released about every year and the established customer base can upgrade to the new version for a small fee. As the product gets older it becomes more difficult to make changes. The new versions are designed so as not to upset old customers by changing operating procedures the users are accustomed to. On the other side, new technology (e.g., the availability of graphical user interfaces) may offer improved opportunities, which are difficult to reconcile with the traditional product architecture. At this critical phase a competing product, which does not have to cope with the past, can take full advantage of the new technology and can triumph. In many cases, the old product fades out and disappears.

Leading software packages are marketed worldwide at comparable prices. For example, we can order a software package directly from a United States wholesale distributor at United States wholesale prices from Vienna via FAX and get delivery the next day via a courier service, effecting payment with a credit card. Such distribution mechanisms force local software companies to sell similar products at matching prices. If a local software company has only a relatively small customer base, it can be difficult to recover the development costs if the product price is determined by "world-market" competitors.

The situation is significantly different if a know-how intensive software package is aimed at a specialized market. In such a situation the marketing activities can be more focussed (and at low-cost) and the competition is less severe.

4.2 Integrated software

"Key-element" software and "intelligent product" software is completely integrated within the product and distributed along the established product marketing channels. This form of software distribution avoids many of the pitfalls of commercial software distribution, such as the problem of illegal duplication, documentation, support, etc.[Kop91]. As the price of computer hardware drops further and further, the strategic importance of the software in an intelligent product increases. Another advantage of integrated software is the generation of additional revenue by the sale of the associated product hardware.

4.3 Free software

In order to avoid the capital expenditures for marketing, advertising and commercial distribution, some authors of software packages decide to distribute their software freely via bulletin boards in computer networks. If a user is satisfied with a program, he is asked to register with the author (or his organization) and pay a small fee for extended documentation, support and future updates. Software distributed in this form is sometimes called shareware. The author of the shareware recovers his costs from the fees paid by satisfied users. He retains the copyright and in some cases restricts the use of the shareware, e.g., it may not be repackaged and sold commercially.

Another form of free software is public domain software, which is free of any copyright restrictions. Public domain software originates often at universities and other research organizations and is freely distributed to the community. The quality and maintenance of public domain software can be a problem.

A thorough report covering many issues relating to free software has been published by UNIDO recently [Bot91].*

5. Software Development Management

In the last few years the techniques for managing software have been further refined. Professional software management is now established in most software companies and quality books about software management are available, such as the practical book by DeMarco *Controlling Software Projects* [DeM82]. In this section we focus on three

* This report is reproduced in Part V of this volume.

software management issues which require special management attention and are sometimes overlooked when setting up a new software production organisation: quality management, risk management and productivity management.

5.1 Quality management

Since many purchasers of software are very concerned about software quality, they start to require that an accredited quality management system is used in the development of the software they are buying.

The International Standards Organization has standardized such a quality management system which is applied to software and other service industries. In its ISO standard 8402 *quality* is defined as the "totality of features and characteristics of a product, process or service that bear on its ability to satisfy stated or implied needs".

Quality attributes

We call those characteristics of the software which are relevant for software quality **attributes**. We distinguish between **functional** and **nonfunctional** quality attributes. Functional attributes are concerned with the mapping from the input domain to the output domain of a software system, i.e., the description of the system functions. Nonfunctional attributes refer to all other requirements not directly related to the systems functions, such as reliability, performance, adherence to development standards, development cost, etc.. In practice, the complete set of requirements is normally in partial conflict. For example, faced with a tradeoff between cost and reliability, the designer is forced to make a decision affecting quality attributes.

Quality can only be managed if the quality attributes are specified precisely in such a form that they can be measured and tested. General quality attributes, such as "the software system must be well-structure", are meaningless. Quality control refers to the operational techniques and activities to ascertain that the stated quality attributes are satisfied.

It has been recognized that quality has to be built in at the point of production and not after the production process at the point of inspection. Therefore modern software development processes integrate quality management with the software production process. In the development of safety critical software, the quality control agencies require the certification of the software production process as well as the certification of the software product.

Quality management system

Quality management starts with the introduction of a quality management system as part of the software production process. The quality management system provides the framework for a coordinated quality policy in an organization. It specifies the strategy and tactics to be followed in order to achieve the intended level of software quality.

In the following section we present an overview of the ISO standardized quality management system (ISO 9001), which is applied to software and other service industries:

(1) Management responsibility: The organization must 'define and document management policy and objectives for and commitment to quality'. In particular, the responsibilities of all staff who perform and verify work affecting quality have to be defined. The documentation of all quality related activities has to be recorded in a corporate quality control manual.

(2) Contract review: Any contract to produce software has to be reviewed from the point of view of quality management. In particular, the quantified quality attributes must be agreed on between purchaser and supplier and must be documented in the contract. Tests as to how to measure the level of quality required must be contained in all contracts. The same applies to software acquired for the project.

(3) Design control: The developer has to establish procedures in order to demonstrate the quality of the design at each design step.

(4) Inspection and testing: Inspection and testing must take place during development and the product status must be recorded at all times. Where appropriate, statistical techniques required to verify the acceptability of product characteristics have to be established.

(5) Quality records: The developer must ensure that sufficient records are maintained to demonstrate that the required quality has been achieved. The quality record keeping must make sure that no unrecorded quality actions took place. Change control must be an integrated activity of the quality management system.

(6) Internal quality audits: The quality control system itself must be subject to periodic reviews to maintain its effectiveness.

(7) Training: Training needs and training levels for all staff involved in the software production process must be specified and recorded.

In some countries, there are organizations to accredit quality control systems. For example, such an organization grants its stamp of approval that the objectives of the ISO 9001 standard are met by a particular quality management system installed in a given organization. In this area of quality management system approval a cooperation between industrialized countries and developing countries could be of benefit to both partners.

Quality control

Whereas the quality management system is concerned with setting up the framework for quality management, quality control is concerned with the operational techniques and activities executed in order to achieve the intended level of quality. i.e., that the quantified quality attributes of the software are met. Quality control is carried out on intermediate and final software products with the intent to uncover weakness in the preceding development process. Quality control can be decomposed in five activities [Oul91]:

1. Define the software quality attribute and its measure.
2. Define the attribute check procedure.
3. Carry out the check procedure.

4. Record the result
5. Take and record any corrective action taken.

It is important that steps 1 and 2 are carried out before the product is ready, i.e., in the requirements analysis phase or in the contract specification phase in contract programming.

If the specification of an intermediate software product is available in a formal notation with formally defined semantics, some properties of the product can be checked mechanically. However, if the intermediate product is not amenable to such automated checks, one has to rely on less formal checking techniques, such as checklists designed to help check for completeness. Birrel and Ould [Bir88] contain extensive check lists for most of the major items produced during software development.

5.2 Risk management

There are many risks involved in software development, which can manifest themselves as technical failures (impaired functionality, poor reliability) and management failures (schedule and cost overruns). Risk management is concerned with the identification, analysis and elimination of these risks before they effect the software project. Risk management involves two steps: risk assessment and risk control [Boe89].

Risk assessment

Risk assessment is concerned with the identification and analysis of the risksassociated with a software project. We distinguish between generic risks, i.e., those that are common to all software development projects and specific risks, which are those that apply only to a particular project.

Risk identification

Risk identification can start with the examination of general checklists for the most common generic and project specific risks. Some of the most important generic risks are:

1. Inadequate personnel
2. Imprecise requirements
3. Unmastered complexity
4. Poor quality

In addition we have to consider such project specific risks, as for example:

1. Imprecise description of the expected work
2. Unrealistic project schedules and budgets
3. Inappropriate staffing lacking know-how base
4. Deficient communication among project members
5. Poor project control
6. Application of unproven technologies

7. Insufficient hardware resources
8. Ill-suited system software
9. Unrealistic performance requirements
10. Unstated or illusory assumptions

These checklists can only serve as a starting point for risk identification. They have to be complemented by a checklist based on local experience.

Risk analysis

Risk analysis starts with the rating of the identified risks in relation to the particular project. This rating has to reckon with the criticality of the identified risk and guess the probability of its manifestation. In some situations it will be necessary to develop an analytical dependability model of the given system in order to assess the significance of a given risk factor. The result of the risk analysis is a weighted list of risk factors relevant to the particular project.

Risk control

Risk control is concerned with the determination of management actions to eliminate, or at least reduce, the identified risks. One first action will be the improvement of the management visibility of those aspects of the project which are prone to risk. This can be done by requiring up-to-date documentation of the achieved progress and by the installation of project management baselines.

If a particular failure mode of a computer system is recognized as very critical, the risk reduction strategy can consist of increasing the resources for verification or the provision of software fault tolerance for a particular function. Furthermore, operational loss-limiting techniques may be installed.

If a high probability of a schedule overrun is suspected, renegotiations with the client are sought in order to modify the completion date or to reduce the functionality that has to be delivered on time.

5.3 Productivity management

In a competitive environment those organizations will thrive which can deliver a given software product of high quality at the lowest cost. **Software productivity**, defined as the relation of software output to the cost of producing this software, is an important parameter of an organization and has to be managed explicitly.

Software productivity can be increased by either reducing the amount of work required to produce a given product or by increasing the effectiveness of the development staff. The analysis of software cost models gives valuable insights into the key factors determining software productivity. Boehm [Boe87] has identified a number of such key factors:

(1) Staff effectiveness: All cost models indicate that the selection, motivation and management of the people involved in a software project are the key productivity factor.

Employing the best people is a good strategy, because the productivity ranges of people are normally much wider than that of their salaries. Continuous training of personnel in technical and managerial matters has a high productivity payoff.

(2) Simple products: During the architecture design phase, every effort has to be made to decompose the system into components that can be designed, implemented and tested independently. The interfaces between components have to be clearly specified and should be free of side effects. If a number of design alternatives to implement a given requirement are available, understandability should be a prime selection criterion. If there are open questions about the implementation of a key software component, rapid prototyping should be considered in order to learn about the difficulties of the solution.

(3) Modern development techniques: The application of modern software development techniques, such as object oriented design techniques combined with an integrated tool support, which covers technical as well as managerial aspects in an integrated fashion, can increase the software productivity and avoid unnecessary clerical rework to accommodate software changes. Automated support tools, such as the provision of an integrated documentation system, can eliminate costly development steps. A good description of the present state-of-the-art in modern development techniques and tool support is contained in [Ald91], where more than ten support environments are discussed. The selection of the most appropriate support environment for a particular organization depends on the type of software produced and the programming methodology/languages chosen. It is a difficult task that requires careful analysis.

(4) Reuse components: The Lines of Code (LOC) which are produced is still considered a reasonable metric for software size. If this size can be reduced by the reuse of software components or the employment of application generators, software productivity is increased. Some software organizations regularly monitor the software reuse factor, i.e., the LOC taken from software libraries and reused relating to the total LOC delivered. Software reuse requires careful management planning. In a "building up" phase standards for software reuse are established and reusable software components identified and classified in a reuse database. In the "design" phase software is selected from this reuse database on the basis of the given requirements profile. There are a number of support systems for software reuse available, which are described in some detail in [Hal91]. Software reuse can be organized on a wider scale than just within a company. If a number of organizations — national or international — agree on a standard reuse database, they can all benefit from such a joint effort.

The level of achieved software productivity is an important measure of success of any commercial software organization. Productivity and quality issues have to be key items on the priority list of top management.

References

[Ald91] A. Alderson; Configuration Management, in: *Software Engineering Reference Handbook*, ed. by J. A. McDermid, London, 1991, pp. 34.1-34.17

[Bir88] N. D. Birrel, M. A. Ould; *A Practical Handbook for Software Development*, Cambridge University Press, 1988

[Boe87] B. W. Boehm; Improving Software Productivity, *IEEE Computer*, September 1987, pp.43-57

[Boe89] Software Risk Management, IEEE Tutorial, IEEE Press, 1989

[Bot91] A. J. A. Bothelho; Emerging issues in the selection and distribution of public domain software for developing countries, Report prepared for UNIDO, Vienna, Austria, 30 April 1991

[Bul91] W. M. Bulekley; Technology, economics and ego conspire to make software difficult to use, *The Wall Street Journal*, Technology section on software, R8, 20 May 1991

[Bus91] *Business Week*, The world's most valuable companies, 15 July 1991, pp. 43-80

[DeM82] T. DeMarco; *Controlling Software Projects*, Yourdon Press, New York, 1982

[Hal91] P. Hall, C. Boldyreff; Software reuse, in: *Software Engineering Reference Handbook*, ed. by J. A. McDermid, London, 1991, pp. 41.1-41.12

[Kop90] H. Kopetz; The production of intelligent products in developing countries, *Microelectronics Monitor*, UNIDO, Vienna, Issue Nr. 29, January 1990, pp. 63 - 71

[Kop91] H. Kopetz, R. Zainlinger, G. Fohler, H. Kantz, P. Puschner, W. Schutz; The design of real-time systems: from specification to implementation and verification, *Software Engineering Journal*, May 1971, pp. 72 - 82

[Oul91] M. A. Ould; Quality Control and Assurance, in: *Software Engineering Reference Handbook*, ed. by J. A. McDermid, London, 1991, pp. 29.1-29.12

[Roo91] P. Rook; Project Planning and Control, in: *Software Engineering Reference Handbook*, ed. by J .A. McDermid, London, 1991, pp. 27.1-27.36

IV

Legal Issues in Software Development, Procurement and Distribution

Recent Trends in Contractual Practice and Jurisprudence Relating to the Licensing and other Forms of Acquisition of Software in the United States of America and the EEC*

Stanislaw Soltysinski**

Introduction

This report contains a succinct review of recent developments in contractual practice, legislation and case law relating to the acquisition of software in the United States and the EEC. References to practice and case law in other jurisdictions (*e.g.* in Japan) have been incorporated in footnotes while discussing parallel problems in American or European law. The section devoted to the EEC concentrates on recent legal developments at the Community level and in Germany. In addition, the report takes into account contractual practice and case law in the United Kingdom and France. Furthermore, the report contains a few references to important judicial and legislative developments in Japan (*e.g.* the issue of legality of "reverse engineering")

Differences between the United States and European laws suggested a parallel analysis of analogous problems constituting the main subject of this study. Thus, for instance, the peculiarities of United States and European rules relating to liability for defective software products dictated somewhat different internal division of the material in the two main segments of the report (Chapter One and Chapter Two). The last chapter is devoted to common problems arising in transnational transactions involving software and computer systems.

In accordance with the terms of the assignment, the report focuses on protection of the recipient (*e.g.* purchaser, licensee, etc.) of software. In contrast to the majority of available commentaries, handbooks and standard contracts, which are usually prepared under the auspices of suppliers of software or hardware, the report is mainly aimed at helping an importer and user of computer programs. While evaluating typical contractual clauses encountered in standard forms used by the computer industry in the United States and the EEC, I was trying to explain their legal consequences and to suggest alternative solutions that could be more advantageous to the recipient of computer technology. However, one must bear in mind that the legal "know-how" constitutes only one prerequisite for a successful deal and it cannot counter-balance disparities in bargaining powers between

* UNIDO/IPCT.139, May 1991

** Professor of Law, University of Poznan, Poland

providers of software and their clients from developing countries. At the same time, an in-depth knowledge of the relevant contractual practice and applicable law is an important factor in negotiations.

A computer program may be acquired and disseminated in a number of ways. The limited scope of this report dictated the need of narrowing it to three main categories of such agreements, namely, "sales", "licenses" and "software development contracts" (commission contract, Werkvetrag). It is further assumed that the foregoing agreements are concluded between computer companies and end-users of software. International distribution and representative agreements are beyond the range of this analysis. In principle, the report is limited to "pure" software transactions but several references are made to case law applicable to "mixed" contracts which cover both software and hardware products.

In accordance with the "job description", the report focuses on three substantive aspects of software acquisition contracts :

1. The scope of the software recipient's right to use the acquired technology,
2. Responsibilities of the supplier for legal and technical defects of software, including tests and guaranties, and
3. The recipient's contractual remedies against the supplier for malfunctioning of defective software.

Special attention is paid to contractual devices aimed at limiting or excluding supplier's liability in the foregoing areas. In addition, the paper scrutinizes problems arising in the context of the so-called "shrink-wrap" licenses and analyzes the dispute concerning the legality of "reverse engineering" (decompiling) of computer programs. Issues common to all transfer of technology transactions, such as fees and other forms of payment, term and termination of the agreement, non-competition and non-assignment clauses, are outside the parameters of this analysis.

The United States

1. General considerations

The difference between "sales" and "licenses" seems to be rather clear. The former contracts consist in transferring the title to "goods" while license agreements are merely permissions to use intangibles by licensees within the limits and on terms agreed upon between the parties to a license contract. In the context of software acquisition contracts this distinction is blurred. First, it is by no means clear whether computer programs are "goods" within the meaning of Art. 2 of the Uniform Commercial Code (UCC), which governs sales transactions in all States, except Louisiana. Second, the owners of software rarely transfer all rights in their intangibles. Normally, they prefer to recoup their investments and profit therefrom by way of granting limited permissions to use the licensed programs to more than one user.

License agreements allow them not only to maximize profits but also to limit potential competition from recipients of their software. Hence, the owner of a computer program usually wants to retain some proprietary rights in order to get access to improvements, control future developments of the disseminated software, etc.[1] When the supplier retains some indicia of ownership in software, the demarcation line between "sales" and "licenses" is not clear. Third, "pure" software transactions are less common than "mixed" contracts involving hardware, software and supporting services.

For reasons stated above, recipients of software should be aware of the implications of the choice between "sale" and "license" form of the transaction. An importer of software, should explore the possibility of "purchasing" rather than obtaining a right to use the target software. Because copyrights and intellectual property rights in trade secrets in a specific computer program constitute a bundle of territorially divisible rights, an importer may sometimes persuade an exporter to sell him software within the borders of the purchaser's country. Similarly, in a border-line case when the supplier of software retains some indicia of title, the recipient should try to insert a clause explaining that the title has passed to the purchaser and the stipulations in favour of the supplier are of purely contractual nature.

Disputes concerning legal characterization of a given transaction ("sale" or "license") are of particular significance with respect to "packaged" software, which is commonly marketed in the "license" form. Courts and commentators agree that it is the content of the legal relationship rather than the denomination of the transaction by the distributor, which should be taken into account in disputable cases.[2]

2. Program licenses with end users

(a) Grant

The term "license" suggests that a licensor is the holder of an exclusive right in the licensed information. It is also presumed that the licensed software is the property of the licensor. The term "property" in this context denotes copyright or proprietary trade secret. Occasionally, a computer program can be protected by a United States patent. Most computer programs are protected concurrently by federal copyright and state trade secrets laws. Such cumulative protection of technology information provides the cheapest and the most effective form of guarding the interests of the licensor. Copyright protection of binary form enhances trade secret protection of underlying non-copyrightable ideas and procedures.[3] And despite obvious conflicting objectives of these two forms of legal protection and notwithstanding the preemption of all legal rights "that are equivalent to any of the exclusive rights that are within the general scope of copyright (...)", the majority of recent precedents shares the opinion according to which a trade secret claim is not **pre-empted** by federal copyright law.[4] And although the purpose of copyright laws has been public access to information and the "dissemination of ideas"[5], at present both courts and the United States Copyright Office reconcile copyright formalities with assertions of trade secrets

rights. By permitting secret deposits of computer programs, the Copyright Office has become an agency collaborating with proprietors of software to bar public access to non-copyrightable ideas (*e. g.* algorithms) indefinitely.[6]

The demise of the supremacy of federal competition law over intellectual property exclusive rights in the present case law, permits the licensor of software to grant both temporary and perpetual licenses. While a copyright license shall not exceed the term of the monopoly right, assertion of trade secrets rights in the licensed software justifies licenses of indefinite duration. Computer industry standard forms indicate that "there is no case or other precedent holding that proprietary or trade secret protection will not be enforced if the license is perpetual."[7] Some commentators advise against perpetual software licenses arguing that such contracts may be treated by courts as "sales" or may be held invalid because of antitrust considerations.

It is clear from the above that suppliers of software are almost completely free to shape the content and scope of their permissions to use licensed software. Apart from time and geographical limitations, software licenses frequently restrict the permissible use to a single computer specified in the agreement by its serial number or in the location designated therein. A typical standard form reads as follows:

"Client shall have the right to use the Licensed Program and materials solely for its own internal operation in the location designated in this Agreement, (or, only in the installation designated in this Agreement , or only on the computer [or machine] designated in this Agreement)"[8]

As an alternative to restricting a grant to a single location or a specified machine, the parties may agree on a uniform company-wide license permitting the licensee to use the program on each of its computers. Large corporations prefer to negotiate for a single license covering all their hardware installations.

Current IBM's "terms and Conditions for Licensed Programs also restrict the use of a program on the designated machine but, in addition, they permit the customer, *inter alia*, to:

"copy or translate the Program's machine readable portion into any machine readable or printed form to provide sufficient copies to support the Customer's authorized Use of the Program; and modify and or merge the Program's machine readable portion with other programs to form an updated work for the Customer's own use.(...)."

In the event the licensee is authorized to use the licensed program only on a single machine, he should try to negotiate a special permission in typical contingency situations involving inoperability of the designated computer or the need to transfer the installation to a new location.[9] Likewise, a prudent licensee should try to negotiate a permission to transfer the licensed programs to another location or to another designated CPU. Bingelow recommends the following standard clause:

"Upon thirty days written notice to Licensor, Licensee may transfer, the Licensed Programs to another Designated CPU on a date specified. Such notice shall specify the date of transfer and thereafter the designated CPU shall be the CPU designated in such notice."[10]

As already mentioned, software licenses are normally limited to the licensee's internal use or operation. This limitation is usually further defined in the agreement. In **Systems Development Corp. v. United States,** [11] the pertinent clause read:

"The National Library of Medicine agrees not to utilize licensed material in the performance of computer service bureau operations nor performance of any services for third parties except within its mission as established by law or regulation."

The court interpreted strictly this ambiguous clause and held that the Library did not violate the terms of the license by providing data to hospitals, universities, etc. Such use of the licensed software was not viewed as providing computer service bureau operations.

An importer of a computer program that contemplates to use the software both in its own company and with the help of third parties is recommended to adapt a flexible clause found in a United States-Canadian licensing agreement:[12]

"The use of the licensed software is to be confined to the Licensee. This License may be exercised by the Licensee by having the Licensor install an operating version of the Licensed Programs on the Licensee's own computer or on such other computer of suitable capacity as may be selected by the Licensee, provided a non-disclosure and confidentiality agreement is first entered into between the owner of such other computer and the Licensor."

Such a clause my be very helpful to an importer from a developing country who wants to benefit from the acquired software without the need to make his own capital investment in more expensive hardware.

A careful licensee should also try to negotiate the right to use the licensed software for the internal requirements of its affiliates. The key terms "use" (or "internal requirements") and "affiliates" require precise definitions in the agreement.[13]

(b) Access to the Source Code; Reverse Engineering, Copies and Adaptations

An important issue in software licensing negotiations is whether the licensor is obliged to provide the licensee with the **source code** and documentation describing the structure or functioning of the program. Licensors prefer to license only an object code and "as little documentation as possible , so that it will be difficult for the licensee to reverse engineer the software to reveal any trade secrets".[14] Following the policy of IBM, the industry leader, the majority of licensors normally expressly prohibit the licensee's access to source codes.[15] Such express contractual prohibitions are aimed at avoiding the impact of the doctrine of "fair use" and other permissive provisions of the Copyright Act (1980).[16]

By contrast, the licensee will be always interested to receive the source code, without which it is practically impossible to modify, enhance or even to remove defects of a licensed program. Licensing a source code is sometimes indispensable. For instance, in an agreement where a software developer licenses a distributor (e.g. a software house or a computer hardware producer), the latter party is usually responsible for continuing obligations consisting in adaptation of the licensed software to the needs of end-users of the product, removing defects and, often, enhancing the original program. But in the context of a license with an end-user, the licensee interested in obtaining access to the source code is usually required to pay an extra fee and accept specific obligations aimed at assuring confidentiality of the code and any information related thereto. Standard contracts dealing with the issue of confidentiality of licensed materials and information embrace both the officers of the licensee and its employees. Bingelow[17] suggests the following formulation:

"Licensee agrees that the source code licensed under this schedule S is the property of and a trade secret of [Licensor]. Licensee agrees not to reveal the source code to any person or entity, except as otherwise provided herein, nor shall Licensee reveal to any other person or entity directly or through its employees any information relating to the source code, the programming therein, or the algorithm thereof."

Some licensors require from licensees that their officers and employees having access to confidential data must execute special confidentiality covenants. Furthermore, licensees are frequently bound to pay agreed damages for breach of such stipulations and the agreed liquidated damages may be higher than the total cost of acquiring the license.

Licensees who receive object codes only, become completely dependent upon the licensor, not only for future enhancements and modifications of the program, but also in the event of software malfunctioning. Access to the source code of a licensed program is crucial in case of importation of foreign software when the distance between the licensee and the licensor makes it difficult to have the licensor's experts available in the country of the situs of the user. Special hazards exists when software is acquired from small software houses that may become bankrupt, thus enabling creditors to attach all unsecured property of the licensor.

A standard compromise solution for such contingencies, consist in appointing an **escrow** agent, with whom the licensor will deposit the source code and its subsequent modifications. Under such an escrow agreement, the licensee has the right to obtain access to the deposited code in the events specified in the agreement, for instance, if the licensor stops maintaining the licensed program, refuses to deliver the modifications thereof or files a petition in bankruptcy. Such an arrangement reduces the risk of destroying secrecy while affording the necessary protection for the licensee.

A standard form elaborated under the auspices of ADAPSO recommends the following language of the Source Program Escrow clause:

"ABC will deposit and maintain with an escrow agent a current copy of the source code of the Licensed Program. In the event ABC ceases to carry on business or ceases to provide maintenence for the Licensed Program, the source code will be provided to client."[18]

In 1985, a bankruptcy decision in which the court permitted a bankrupt licensor to reject its nonexclusive license to a licensee, was interpreted as a precedent that could preclude the usefulness of the escrow agreements.[19] Soon afterwards, the United States Congress has passed a corrective legislation, which reiterates the validity of escrow rights in the field of intellectual property. The Intellectual Property Bankruptcy Protection Act states that if a bankruptcy trustee "rejects" a license, then the licensee may either treat the rejection as a termination and submit a claim for a breach of contract, or retain the license, continuing to make royalty payments to the estate. The trustee must not interfere with the licensee's right to obtain copies of intellectual property technologies (*e.g.* a source code), but there is no obligation of maintenence or support on the part of the bankrupt estate in such a situation.[20]

Industry standard forms for software licenses with end users often contain express prohibitions on **reverse engineering, modifications and adaptations** of software. Reverse engineering is the process by which a product embodying an innovation is analyzed by a competitor in order to study or to reproduce thereof for competitive purposes. While such inspection is perfectly legal in all other areas of intellectual property,[21] hybrid protection of software under trade secret and copyright laws enables the software developers to argue that decompilation and disassembling of computer programs amount to an illegal copying of the protected form of software. Since the law in this field is not settled, suppliers of software can afford to preclude reverse engineering and modifications of the licensed software by inserting express contractual prohibitions into their agreements with licensees.

It is interesting to note that prior to its 1983 announcement,[22] IBM apparently tolerated decompilation and adaptation of its software. Since then, however, the company's ban on reverse engineering has become an industry legal standard. "Agreement for IBM Licensed Programs" provides that a licensee (customer) shall not reverse, assemble or decompile in whole or in part any licensed program.[23]

While judicial authorities and commentators are divided on the issue of legality of "reverse engineering" in the area of "copyrighted" software, the licensors of computer programs may also rely on contractual bans enforceable under state laws. While CONTU Report[24] and some recent cases[25] suggests that it may be a "fair use" under Section 107 of the Copyright Act to make a printout copy of software for purposes of making modifications and other purposes, there is no judicial authority questioning the validity of contractual prohibitions on reverse engineering in arms-length licenses.

As a rule, software license agreements with end users strictly control the number of **permissible copies** of each licensed program and other proprietary materials supplied to the licensee. Restrictions on the number of authorized copies the licensee can make during the agreement cover not only source codes but also object codes, manuals, instructions, etc. The

Licensee is permitted to make one copy of each program for back-up purposes **(archival copies)**. In the event of loss or accidental destruction, he may obtain an extra copy subject to an additional payment. Frequently, licensees are obliged to observe special measures aimed at assuring compliance with confidentiality obligations:

*"(a) **COPIES.** As provided in Clause 3(c), Licensee may make copies of the Licensed Software, provided that each such copy shall state that it is the property of Licensor , ..., in the following language:*

"this copy of....[insert the name of program or manual) is the property of [insert the name of the Licensor] ...., as their interests may appear and is protected under the copyright, trade secret and confidentiality laws of the United States and Canada. At Licensee's request, Licensor will provide a label to be attached to the copy setting forth the foregoing statement. Licensee shall keep a record of each copy made, where such copy is located and in whose custody it is. The provisions of this clause shall apply to all licensed Software, including without limitation programs, manuals, instructional materials and all other documentation provided to Licensee."[26]

Practical significance of such contractual steps to protect the confidentiality of the information licensed cannot be overestimated. The problem is illustrated by **Data General Corporation V. Digital Computer Controls, Inc**.[27] The plaintiff alleged misappropriation of its proprietary information embodied in n. ntenance drawings for a Nova 1200 minicomputer, which had the following legend:

"These drawings and specifications, herein, are the property of Data General Corporation and shall not be reproduced or copied or used in whole or in part as the basis for manufacture or sale of the items without written permission."

It was rather surprising, even in the context of the present generous attitude of the United States judiciary vis-a-vis propriertors of trade secrets, that the court ruled for the plaintiff despite the fact that the pertinent maintenance drawings had been made available to about six thousand people and the mass distribution of the alleged secrets by the proprietor did not destroy its claim for trade secret protection.

(c) Tests, Training, Maintenance and Enhancements

As explained in IBM's "Terms and Conditions for Licensed Programs", the purpose of a **trial period** is to allow the customer to determine that an acquired program meets its requirements. A trial period may be either a **Testing Period** or a **Return Period.** Where the former applies:

"The Customer may use the Program only for non-productive purposes during this period to determine that it meets its requirements.(...) The Customer may terminate the License upon written notice, effective immediately, at any time, during the testing period, in which event [charges specified in the Agreement] will not be due. However, process charges, if any, will be due."

Where a **return period** applies, the licensee may return the licensed program but it must end the agreement by written notice because:

"Unless such notice of termination is given, the customer will be deemed at the end of the trial period to retain the Licensed Program under the provisions thereof."

According to a survey conducted by ADAPSO among its members, an agreement between a software company and an end user is the most frequent type of licensing contracts. Although some software licenses and industry standard forms characterize "program support services" as optional, according to the same source, such services are usually included with the license at no extra cost.[28] Among the services most frequently offered free of charge are installation, training or maintenance. A license contract will usually specify the scope of such additional services to be rendered at no extra charge during the term of the agreement. The following sample clauses illustrate the practice:[29]

"[Licensor] will provide up to ... days of training (or, operator instruction to ...[number of persons] designated by [Licensee]) in the use of the licensed Program and Materials (on Client's computer equipment."

"[Licensor] will provide maintenance of the Licensed Program for a period of.... month(s) or, year(s)) from the date of delivery (or date of installation) of the Program (and Materials)."

"Upon written request, Licensee will provide, for a period of month(s) (year(s)) after execution of this Agreement (or, after delivery of installation of the Licensed Program and Materials) enhancements to the Licensed Program [and Materials] that are marketed by Licensor."

Characteristically, software industry standard contracts often provide clauses to the effect that any unauthorized enhancement to a licensed program by the licensee deprives him of the benefit of program support services. **"Software product maintenance and support agreements"** frequently constitute a separate part of a software acquisition contract. In such cases, they provide for additional fees to be paid by the licensee. Sometimes, suppliers of comprehensive computer systems covering the delivery of software and hardware insist on executing a legally autonomous maintenance and support contract that becomes effective after the installation of the system. The separation of a basic software acquisition contract from its functionally related "maintenance and support" agreement may be dictated by various factors (e.g. tax considerations, requirements of foreign technology control laws, inability of the main supplier to undertake extensive support obligations, etc.).

The licensee may prefer to enter into a separate software program maintenance agreement with a third party in the event when the recipient of technology is in direct competition with the licensor and does not want its commercial secrets to be disclosed to the personnel of the competitor. Similarly, the licensee may prefer to order installation and training services from the actual developer of a licensed program, who is more familiar with the technology than the licensor, who acquired the title to the licensed software from the programmer.

The **maintenance and support** agreements usually mean removal of programming errors, maintaining a licensed program operational in conformity with specifications, supplying the licensee with updated user guidelines and updates to the licensed programs. In addition, the licensee may be also authorized to request "**enhancements**" of the original software. Some sample contracts carefully distinguish between "**updates**" and "**enhancements**". [30] Because some program maintenance and support obligations overlap with warranty obligations (*e.g.* correction of errors), some aspects of a licensor's duty to remove defects of software will be discussed in Section 2.5 of this report.

(d) Warranties of Title

In view of the uncertainties associated with the legal status of software programs, licensors are reluctant to grant express warranties of title and tend to limit their potential liability by way of contractual disclaimers.[31] Foreign licensees should be aware that standard warranties of property rights employed by American licensors offer them a minimum of protection and should not be confused with comprehensive warranties of title. Thus, for instance, a promise that licensor "warrants that it has the right to grant a license to the licensed program", may be interpreted merely as an assurance that the licensor has obtained the authority to license from a third party, for instance, the developer of the licensed computer program. Such clauses are used also by program developers who wish "to guarantee what is minimally necessary to ensure the legality of the license grant."[32] Therefore, the licensee is advised to check the validity of the licensor's authority or to demand a stronger legal guarantee to title ostensibly owned by the supplier.

A warranty of software development by the licensor is also a very weak form of guarantee. It simply means that the licensed program was conceived by the licensor ,but the latter does not guarantee that the software does not infringe third party intellectual property rights. Therefore, a licensee with a strong bargaining power or a "deep pocket" may try to obtain both an express guarantee of title and indemnity in the event of infringement of third party rights. Such a double protection scheme can be drafted along the following lines:

"The Licensor warrants that it is the sole owner of the licensed software that is free of any third party rights (e.g. liens, encumbrances, etc.). The licensor further warrants that, to the best of its knowledge, its proprietary rights are not challenged or disputed by any third party. In the event of a claim that the use of the licensed program constitutes an infringement of a third party right, the licensor will indemnify the licensee from all direct and consequential damages."

Realistically, however, the chances of obtaining such protection by a licensee are often marginal. A sensible compromise may consist in combining an assurance that the licensed program does not violate third party rights (to the best of licensor's knowledge) with a promise to defend licensee in the event of a challenge by such parties. Consider the following sample clauses prepared under the auspices of ADAPSO :

"In the event of a copyright or patent infringement claim, [Licensor] may at its own expense defend such claim or may procure the right to continue using all or part of the Licensed Program or may discontinue the Licensed Program. This shall constitute the entire liability of [Licensor] with respect to a copyright or patent infringement claim."[33]

"In the event of a claim that the Licensed Program constitutes an infringement of a copyright or patent, [Licensor] will indemnify [Licensee] from direct expenditures incurred by [it] in defense against such claim, provided that [Licensor], in its judgement, shall receive the cooperation and assistance of [Licensee]."[34]

Note that in both instances, the licensors' liability is limited to patent and copyright infringement suits while a license can be attacked for violation of proprietary information. Second, the first clause gives the licensor a full discretion whether to defend a third party challenge.In the second sample, the licensor may avoid liability arguing that the licensee has failed to provide him sufficient assistance. Therefore, a judicious licensee should be able to distinguish between limited but meaningful warranties from sham and discretionary assurances. Therefore, for instance , the parties should agree on the scope of cooperation while defending third party suits.

Some major computer firms, notably IBM, grant their customers very effective warranties of title. Standard Terms and Conditions used by that company read as follows:

"IBM will at its own expense settle or defend, and pay any damages or costs resulting from, any claim brought against the customer that any machine, program package or programming or the use of any material within the scope of a License or any use thereof infringes or has infringed a patent, design right ..., moral rights copyright or any intellectual property right effective in the United Kingdom provided that the customer.

i) promptly notifies IBM in writing of any such claim; and
ii) permits IBM to control the defence and settlement of any such claim."

(e) Warranties of Fitness and Merchantability

Implied warranties of **fitness** for a particular purpose and warranties of **merchantability** of software roughly correspond to statutory liability for "physical" defects of goods in civil law countries. Like in Europe and Latin American countries, the threshold problem is whether code provisions governing the seller's liability for defective performance are applicable to transactions involving software. Two arguments are advanced against the application of Art. 2 of the Uniform Commercial Code on Sales to licensing of software. First, software is

not a "good" within the meaning of & 2-105 because software is an intangible. Second, software licenses are not "sales" because the title to the intangible remains with the licensor.[35]

Objections against applying traditional sales law concepts have gradually faded away. Today, the majority of commentators agree that the UCC provisions on warranties should govern either directly or by way of analogy the obligations of suppliers of software *vis-a-vis* their clients. A recently published handbook concludes, that Art. 2 of the UCC governs most computer-related transactions, except those that can be characterized as contracts for rendering solely services.[36] Judges and parties to software legal disputes tend to rely on the Code, because the standards established in Art.2 constitute the only comprehensive codification of warranties in business transactions. Efforts to pass special legislation devoted to the liability of suppliers of software have failed due to the lobbying of the industry.[37]

Since courts and legal commentators tend essentially to treat various forms of transactions in software equally and in reality, software licenses are often merged with transactions for the delivery of hardware and services (*e.g.* computer system agreements or "turn-key" computer contracts), a more comprehensive description of legal consequences of the subjection of the said computer software deals to Art. 2 of the UCC, is presented in Section 4 (infra). Below, we will briefly present legal standards prevailing in the United States licensing practice.

According to an industry survey, the warranty clauses belong to the most frequently negotiated contractual stipulations. Yet, at the same time, the content of warranty clauses used by members of ADAPSO reveals that these clauses are strikingly similar. It proves that the clauses are "so to speak non-negotiabl.[38] A review of warranty stipulations used by the industry leaders and recommended by leading software law handbooks indicates that with the sole exception of **custom-made programs,** licensees can rarely count on obtaining a bare minimum of guarantee that a licensed program either fits for a particular purpose or that its operation will be substantially error-free. For example, a program license agreement for IBM's 2.1 DOS reads:

"LIMITED WARRANTY: THE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE YOU ... ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION ... IBM does not warrant that the functions contained in the program will be uninterrupted or error-free. However, IBM warrants the diskette(s) or other medium on which the program is furnished, to be free from defects in materials and workmanship under normal use for a period of 90 (ninety) days from the date of delivery to you... LIMITATIONS OF REMEDIES: IBM's entire liability and your exclusive remedy shall be: the replacement of any diskette ...2. if IBM or the dealer is unable to deliver a replacement... which is free of defects .. you may terminate this agreement."[39]

The foregoing contractual provision, which follows Art. 2 requirements concerning warranty exculpation clauses, contains the notorious "**as is**" disclaimer excluding all implied warranties. According to & 2-316(3)(b) of the Code, such expressions like "**as is**" or "**with all faults**" exclude all implied warranties, which means that the buyer (licensee) takes the entire risk as to the quality of the goods. Foreign importers of software should also be aware that by examining the computer system or refusing to examine the licensed program, they may lose implied warranties "with regard to defects which an examination ought in the circumstances to have revealed to [them]" (& 2-316{3}{b}).

Against this backdrop, licensees ought to try to negotiate a meaningful protection with respect to the reliability of program performance and its fitness to a concrete purpose. The two sample clauses reproduced below seem to represent a compromise approach:

"Licensor warrants that the Licensed Program shall perform in accordance with the specifications set forth in the licensed materials and fits the purpose described in the 'whereas clause'".

"Licensor warrants that the licensed program will perform substantially in the manner described in the licensed materials if it is properly used as described in the instructions and manuals delivered to Licensee."

It is worth noting that the latter clause contains two important qualifications in favour of the licensor. The word "substantially" takes into account the legitimate interests of the licensor. Indeed, there are no perfect error-free programs and they are constantly modified. Therefore, parties to a licensing agreement should negotiate a program of long-term cooperation aimed at both removing defects and perfecting the licensed software.[40]

A discerning licensee should describe in detail the intended use of the computer program to be acquired and obtain precise information from the licensor concerning the advantages of his software, its compatibility with the user's hardware, etc. The licensor's explanations, especially given in business correspondence, may be classified as "**express warranties**". Licensors, however, try to disclaim liability for inducing their clients to enter into contracts by relying on the **parol** provision of & 2-202 of the UCC, **disclaimers** and the so-called **merger** clauses. Consider, for instance, the disclaimer and merger clause in **Investors Premium Corp. v. Burroughs Corp.**:[41]

"There are no understandings, agreements, representations or warranties, express or implied (including any regarding merchantability or fitness for particular purpose) not specified herein, respecting this contract or the equipment hereunder. This contract states the entire obligation of the seller in connection with this transaction."

The purpose of such a merger clause is to negate an express warranty, usually given before execution of the contract but, sometimes, it is used to disclaim even a warranty given after signing the agreement. Commentators stress the fact that the majority of United States courts usually approve the validity of such disclaimers embodied in merger clauses.[42] The

first precedent in the computer transaction field that explicitly rejected the proposition that disclaimers are effective against express warranties was **Consolidated Data Terminals v. Applied Digital Data Systems.**[43] Although those cases dealt with computer hardware and computer systems, it is likely that their holdings are applicable to "**mixed**" (software-hardware) and "**pure**" software transactions.

3. Shrink-wrap licenses

The development of personal computers, combined with the ability of software firms to distribute programs on floppy disks, have enabled them to market software in small packages.Typically, such programs are recorded on diskettes and marketed in plastic envelopes containing both software and instructional material. Although the distribution of programs is analogous to selling books, software companies have chosen a **license form** of contracts in order to protect their trade secrets and control the market.The advantages of the license approach are manifold: first, software owners are able to retain title to the program and the medium in which the software is recorded in. Second, they can unilaterally delineate the scope of the client's right to use the program and disclaim all warranties. Third, prhaps they can even defeat those provisions of the Copyright Act, which are aimed at securing the owner of a copy of a program the right to make **adaptations** and the right to make a **back-up copy.**The retention of title enables the owner of software to prevent reverse engineering and avoid the application of the First Sale Doctrine that prohibits the copyright owner to control copies of books and other works which were sold in a market place.

"**Shrink-wrap**" licenses, also known as "**blister-pack**" or "**box top**" licenses, are typical **contracts of adhesion**. Unlike arm's-length license agreements, they leave no room for negotiations and, according to their terms, the contract is made upon the opening of the package containing a program by the client. An offer to clients printed on the wrapper typically reads as follows:

"BEFORE YOU OPEN THIS PACKAGE: CAREFULLY READ THE FOLLOWING LEGAL AGREEMENT REGARDING YOUR USE OF THE ENCLOSED PROGRAM. OPENING THIS PACKAGE MEANS YOU ACCEPT THE TERMS AND CONDITIONS OF THIS LICENSE. IF YOU DO NOT AGREE WITH THEM, YOU SHOULD RETURN THE PACKAGE UNOPENED AND YOUR MONEY WILL BE REFUNDED"

Some software companies try to obtain the customer's signature by providing a registration certificate, which may allow the "licensee" to claim his money back or enforce skeleton warranties but many software "publishers" do not bother to obtain such evidence of the client's assent. Typical terms of shrink-wrap licenses used by suppliers of such popular programs as Lotus 1-2-3, WordPerfect, Apple Writer usually contain the following terms and conditions:

1. The geographical scope of the license is limited to the territories of the United States and its possessions;
2. The licensee is permitted to use the program on a single machine;

3. The transfer of the program is prohibited or conditioned upon the acceptance of the terms of the license by the sublicensee;
4. Reverse engineering and adaptations of the program are usually prohibited;[44]
5. Programs are normally licensed on "**AS IS**" basis, that is to say "without warranty of any kind";[45]
6. Licensors disclaim any damages but, sometimes, licensees are allowed to claim the replacement of a defective medium (e.g. diskette) or they may obtain a refund upon returning the program if the licensor or its dealer is unable to deliver a replacement;
7. To further protect the licensor and its dealer against the risk of express warranties arising during negotiations or discussions preceding the agreement, shrink-wrap licenses contain "**merger**" or "**entire contract clauses**" which state that the licensee acknowledge that he has read the contract and agrees that the agreement "supersedes any proposal or prior agreement, oral or written, and any other communications between [the Parties] relating to the subject matter (...)."[46]

The phenomenon of shrink-wrap licenses has been analyzed in numerous law reviews and many commentators stress the fact that they epitomize all vices of contracts of **adhesion** allowing the party of superior bargaining strength to dictate its terms unilaterally to its clients.[47] Recognizing that courts may refuse to enforce such licenses either as a matter of contract law (the lack of assent of the purported licensee) or because their terms are inconsistent with federal laws, the software industry has been lobbying several state legislatures to enact laws validating shrink-wrap transactions. In 1984, the first statute of such nature went into effect in Louisiana.[48] The enactment provides that an acquirer of mass-marketed software enters into a license agreement on terms formulated by the supplier upon opening a package. The Louisiana Law validates the following terms of the license:

a) Stipulations for the retention of title to the software by the licensor;

b) Prohibitions against "reverse engineering", copying, modifying and adapting the software;

c) Prohibitions against assignment, rental or other disposition of the software; and

d) Stipulations for the automatic termination of the agreement if any license term or condition is breached by the licensee.

The Louisiana Statute was challenged soon after its entry into force. A district court decision held that several of its provisions are preempted by the federal Copyright Act and the ruling has been upheld by the court of appeal.[49] The court held that the Louisiana Software License Enforcement Act created a perpetual prohibition "against copying any computer program licensed pursuant to its provisions".[50] Thus it clashed with the archival copies exemption of Section 117(2) of the Copyright Act, which was established by Congress for users of software. Furthermore, the opinion states that the Statute has touched upon the area of the federal patent and copyright laws by permitting a software producer to prohibit the adaptation of a licensed program by reverse engineering.

Vault raises an interesting and controversial question. To what extent, if at all, does its rationale apply to normal licensing agreements that are not contracts of adhesion? The argument that the conflict between state laws enforcing covenants prohibiting all forms of reverse engineering and adaptations and federal laws propagating dissemination of ideas and competition should be resolved in favor of the latter laws, is consistent with some United States Supreme Court decisions, but there are also precedents established by state and federal courts enforcing perpetual prohibitions on licensees of trade secrets that are justified in the name of the parties' autonomy and freedom of contract.[51]

4. Sales and computer system agreements with end users

As already mentioned, "pure" software transactions take the form of "license" or "lease" to enable the supplier to retain the title to the marketed intangible. And as long as the rationale of **Vault** is not extended by courts to normal contracts which are actually negotiated by users of software, licensors will be able to circumvent & 117 of the Copyright Act, which authorizes the owner of a copy of a computer program to make or to authorize the making of another **copy** or **adaptation** of that computer without the permission of the copyright owner.[52] Because the statutory privileges are granted to "owners", instead of "lawful possessors"[53] of copies of programs, licensors and lessors of computer programs forbid their clients to exercise their rights by inserting appropriate contractual prohibitions into their agreements.

For the foregoing reasons, acquisition of software in the United States and elswhere is typically conducted through licenses (leases) rather than through various forms of sales-like transactions.[54] The latter arrangements typically comprise the delivery of hardware, software or services. The majority of reported cases deal with such mixed transactions involving the sale of hardware, including its system software, combined with the sale or licensing(leasing) of application software needed by the purchaser. In addition the transaction may cover the provision of maintenance and support services. These complex contracts are usually denominated as "**computer system**" or "**turnkey**" agreements.[55]

A review of the recent case law confirms the proposition that courts apply Art. 2 of the UCC to the overwhelming majority of software acquisition contracts either directly or by way of analogy. The Code provisions on "sales" govern not only mixed transactions involving software and hardware but also the lease-purchase of computer software.[56] **Customized** computer programs have been also classified as "goods" for the sole purpose of allowing the court to apply Art. 2 of the UCC to such transactions.[57]

The application of Art. 2 of the Code to the majority of computer-related transactions means, inter alia, that the following practical aspects of such agreements are governed by the proper state law on sales:

1. Warranties,
2. Remedies for breach,

3. Disclaimers and limitation of remedies,
4. Mergers and integration clauses,
5. Statute of limitations periods and
6. "Vouching in" rules.[58]

We have discussed some of these issues in the context of software licensing agreements (*supra.* items 2 and 3). To avoid repetition, this section of the report will examine only selected new issues against the background of recent court decisions in software **sales** transactions.

The Code creates an implied **warranty of merchantability** (&2-314), which requires that the product be of reasonable quality and fit for its ordinary uses. Some commentators argue that the standard of "merchantability" cannot meaningfully be applied to software transactions because computer software are so diverse that they cannot be properly described for the purpose of defining their quality characteristics or minimum functions the program can be expected to perform.[59] Although this statement seems to be disputable, it is interesting to observe that there are very few court decisions in which software vendees prevailed in a suit for breach of warranty of merchantability. Similarly, software vendors frequently prevail in actions involving allegations of breach of warranty of **fitness for a particular purpose** (& 2-315 of the UCC). A case-by-case analysis of UCC software litigation reveals that vendors successfully rely on warranty **disclaimers** not only in **implied** but also in **express** warranty disputes. The latter form of warranties can be created both by agreement and any affirmation of fact or promise made by the seller that constitutes part of the bargain (&-313 of the Code) . Typically, any description of the quality or function of a computer program may become classified as an express warranty (*e. g.* a representation that a program has an on-line response time or that a system is adaptable for a specific type of business).

While in "normal" sales transactions, courts have generally not allowed vendors to rely on disclaimers to disavow their written or oral **express warranties**, in the majority of computer-related cases, they have reached the opposite conclusion.[60] Thus, for instance,in **Westfield Chemical Corp.v. Burroughs Corp.** the court held that an express assurance concerning the time saving of a computerized accounting system was effectively disclaimed.[61] Similarly, in **Jaskey Financing and Leasing v.Display Data Corp.** the court ruled that a conspicuous and properly worded disclaimer, effective against **implied warranties,** precluded oral **express** representations that the sold software system was adaptable to the buyer's type of business.[62]

The minority view exemplified by **Consolidated Data Terminals v. Applied Digital Data Systems**[63] rejected the seller's proposition that the disclaimer of "express and implied warranties" should override an express representation relating to the operating speed of a line of terminals sold to the purchaser.The author of a law review lists, *inter alia*, the following obstacles facing purchasers of computer systems under United States law:

1. The seller's ability to limit or disclaim warranties,
2. Limitation of consequential damages
3. The buyer's failure to effectively reject defective goods or to particularize the defects, and
4. The shortening of the statute of limitations.[64]

Industry surveys indicate that "sales" of computer systems are subject to the same kind of warranty standards as licenses and leases of software. Typically, suppliers of software grant limited warranties ranging from 30 to 120 days of duration. Suppliers of custom-made software usually give longer warranties. One year warranties are not uncommon.[65]

In practice, the seller's obligations are usually reduced to "**fix or replace**" duty. The following language is recommended by an ADAPSO sample form:

"If ABC is unable to replace defective documentation or media or if ABC is unable to provide a corrected computer program or corrected documentation, ABC will at its sole and exclusive option either replace the computer program with a functionally equivalent program or refund the fees paid for licensing the computer program without charge."[66]

Legal analysts emphasize that until recently courts were decisively biased in favour of the suppliers of computer programs because of the "infant industry" argument. At present, many legal analysts, consumers and even the software industry recognize the need to enhance the protection of users and condone misrepresentation by suppliers of computer programs. The software industry, without conceding the issue, is advising its minicomputer segment to take into account the potential applicability of The Magnuson-Moss Warranty Act[67] to microsoftware transactions. The Act applies to "consumer products" and imposes certain mandatory obligations on sellers of such merchandise.

To sum up, United States laws governing the acquisition of software is still biased in favour of the seller and, therefore, foreign importers of United States software should be aware of many potential legal traps associated with the application of the UCC to their transactions. In arm's-length transactions, if they cannot agree on the choice of a more fair legal system, they should seek advice of an expert in the law of the exporter.

Recent Developments in the EEC

1. Introduction

As indicated, the second part of the report is devoted to a succinct review of recent legal developments in the EEC. The evolution of national laws is illustrated mainly with references to jurisprudence and practice in the Germany. Germany has the richest collection of judicial precedents in our field. Moreover, the dominant trend of the case law in that country is rather well-balanced. By and large, German courts take into account reasonable

expectations and justified interests of both the suppliers of software and users thereof. This point seemed to be relevant from the perspective of the Regional Network for Microelectronics in the Economic Commission for Latin America and Caribbean (ECLAC) region.

Importers and users of technology from Latin American and Caribbean countries are advised to examine and adapt various European model laws and general conditions for the acquisition of computer programs. The following sections of the report contain numerous references to the Conditions for Supply of Licensed Software Packages to Government Users in the United Kingdom (the United Kingdom Government Procurement Conditions 1987) and similar general conditions being in force in Germany.[68] Those standard forms and general conditions establish *de facto* legal standards and are more even-handed than contractual forms elaborated by the software industry.

2.Licensing of software and computer systems

(a) General Considerations

Unlike the United States or the United Kingdom, where apart from codification of sales laws, the remaining contracts are almost exclusively the domain of general law of contract, in Germany, as in many civil law countries, the Civil Code (BGB) regulates a number of typical contractual transactions (*contracti nominati*). As a result, classification of a given transaction under a specific rubric (*e.g.* "sale", "barter", "lease", etc.), entails important practical consequences for the parties thereto. Specific types of contracs are subject to different standards of form, remedies, statute of limitations, etc. Transactions not regulated by the legislator, are governed by general rules applicable to all contracts. Besides, such unnamed contracts can be governed by the provisions applicable to similar types of contracts regulated in the Code by way of analogy.

Intellectual property licenses are not regulated in the Civil Code. Thus, in principle, they are subject to its general rules on contractual obligations. According to the German Supreme Court, contracts for the supply of software — depending upon their content — qualify as lease (*Miete*), leasing, usufructuary lease, contract for work or sale.[69] Although the issue of classification of software licenses, has not been finally settled, the dominant view in the jurisprudence is that they are governed by general rules of the Code applicable to contractual obligations.[70] Some commentators advocate the application of the pertinent provisions relating to lease (*Miete*) or usufructuary lease (*Pacht*).[71] As explained by the Landsgericht Stuttgart[72], the application of the general rules gives the licensor the right to cure defects (& 326 of the Code), but extends the statute of limitation period from six months to 30 years.

In principle, software licenses are limited to situations when the marketed data are copyrightable or constitute trade secrets and when the supplier retains the title.

(b) The Scope of the License

Both in Germany and other EEC countries, suppliers of software are essentially free to delineate the scope of the grant. Territorial, time and subject-matter restrictions are permissible within the limits permissible under the respective national and Community antitrust rules. As in the United States, licensees are often expressly prohibited to copy the licensed program, except for a back-up copy, and may not adapt it for purposes not contemplated in the agreement. Furthermore, the licensee may not **rent** the licensed program to third parties.

(c) Training, Maintenance and Support

Neither United Kingdom nor German laws of contract imply a general obligation of the licensor to provide the licensee with free of charge training and support services.However, German courts have developed a principle that the licensor owes the licensee a duty of advice concerning the choice of the best combination of software and hardware for a concrete purpose described by the client.[73]

Model Form of License Agreement for the Use of Computer Software Products, elaborated in the United Kingdom under the auspices of the Institute of Purchasing and Supply (IPS Model L, 1987), stipulates that the licensor, shall "if applicable , install the program by the date, all as specified in App. 1". Also, maintenence and support obligations arise only "if required" (Clause 11). On the other hand, the IPS Model License is more generous to the licensee with respect to the issue of **training:**

"The licensor shall provide instruction in the use of the Program for the Licensee's personnel as specified in App.1. Unless otherwise specified no charge shall be made for such instruction but the Licensee shall be responsible for paying any travel or living expenses."

(d) Responsibility for Legal Defects (Warranties of Title)

Analogous application of sales concepts or rules applicable to intellectual property licenses under the majority cf European legal systems would lead to the imposition of sanctions upon the licensor in the event of "legal defects". Therefore many software suppliers and some commentators argue that far-reaching limitations of licensor's liability should not be objectionable. Consider the following sample form:

"The Licensor is not aware of any rights of third parties which would oppose the utilization purposes of the Licensee. The Licensor is not liable, however, for the licensed software being free of rights of third parties."

"If the Licensee is accused by third parties of infringing intellectual property rights (...), the Licensor promises to provide the Licensee with information and documents in defense against such claims as far as the licensor is able to do so without breach of third party obligations and while maintaining its own confidentiality interests. All costs involved in such activities shall be borne by the licensee."[74]

In contrast, the IPS Model provides that the licensor shall fully indemnify the licensee against all damages (excluding **consequential** damages) incurred by reason of any infringement or alleged infringement of the licensed program subject to the following conditions:

i) "*The Licensee shall promptly notify the Licensor in writing of any alleged infringement of which he has notice;*
ii) *The Licensee must make no admissions without the Licensor's prior written consent,*
iii) *The Licensee, at the Licensor's request and expense shall allow the Licensor to conduct any negotiations or litigation and or settle any claim. The Licensee shall give the Licensor all reasonable assistance. The costs incurred in such negotiations shall be for the Licensor's account.*"

A similar copyright indemnity clause is found in the Central Computer and Telecommunications Agency and Computing Services Association Software License General Conditions (CCTA & CSA Licensing Conditions). This proves that it is a myth that a software supplier cannot undertake an effective copyright indemnity obligation.

(e) Responsibility for Physical Defects — Warranties of Merchantability and Fitness for a Particular Purpose.

Although the European computer industry also stresses the point that software is never free of errors, the jurisprudence in Germany and other EEC countries is less supplier-biased than the United States case law. In Germany, for instance, the judicial concept of "physical" defects of software is much broader than in United States case law and, surprisingly, there are numerous reported cases devoted to warranty disputes there. The notion of "defect" is defined in & 459 of the Civil Code in the chapter on Sales but it is applicable to defective performance within the framework of other "named" contracts such as "lease" or "work". Since software licenses are classified usually as "lease-like transactions", we will limit our analysis to a brief examination of lessee's remedies in case of breach of warranty.[75]

According to & 537 of the Civil Code, the lessee may either suspend paying rent (in the event the product is inappropriate for the agreed use) or reduce its payment (in case the product has a defect limiting its usefulness for the agreed use). In addition, if the lessor fails to remove defects in due time, the lessee (licensee) has the right to sue for damages. In addition, the licensee may remove defects at the licensor's cost (&538). Finally, the lessee (licensee) may rescind the contract if the lessor (licensor) failed to make the thing good within a reasonable cure period (& 542).

General conditions and sample contractual clauses frequently limit the licensee's right for breach of warranty. Major computer companies offer a full range of warranty terms. Siemens' General Terms of Software Licenses, for instance, offer three classes of warranty obligations that are subject to different price conditions:

a) Full warranty, including elimination of errors;

b) Partial warranty plus assistance in the elimination of errors; and

c) Replacement of defective software and/or assistance.[76]

In the United Kingdom, the licensor of software typically warrants that the licensed program will perform substantially as described by the enclosed technical documentation. A contractual warranty found in the CCTA & CSA general conditions promises that:

"Unless otherwise provided in the Special Conditions, the Licensor warrants that the Product used in accordance with the Licensor's instructions will perform substantially in accordance with the operating manual ... for the duration of the Warranty Period specified in the Schedule. The Licensor does not warrant that the functions or facilities of the Product will meet the Licensee's requirements or that operation of the Product will be uninterrupted or error free."

Similar terms of warranties are recommended by the Model Form of License Agreement of IPS.

(f) Disclaimers

European standard forms and general conditions often expressly exclude the licensor's liability for consequential losses. Licensors tend to disclaim their liability for warranty of fitness. Some model forms recommend the language stating that the licensee knows the licensed program and its technical capabilities.[77] However, some model agreements provide that "the licensor shall indemnify the licensee against injury to any persons or loss of or damage to any property, including the program which may arise out of default or negligence of the licensor."[78] The CCTA & CSA Conditions contain a similar provision, qualified by a ceiling of the upper amount of any liability in respect of losses and damages to property up to the amount of one hundred thousand pounds.

IBM's Standard Terms and Conditions used in the United Kingdom offer warranty of conformity of the licensed program with the current specifications: "All other licensed programs are distributed "**as is**", without warranty of any kind, express or implied." But in the event the company is found liable for death, personal injury or for damage to tangible property its liability is unlimited. In other cases, the conditions specify contractual maximums of liability. For instance, the upper limit of compensation to be paid for a licensed program, IBM promises to pay the greater of £55, 000 or twelve months' charges due for the use of the licensed software.[79]

(g) Shrink-Wrap Licenses

As in the United States, "shrink-wrap" licenses are widely used in Europe. They are marketed on similar terms and conditions. Thus, for instance, standard software packages offered by Image System Technology, United Kingdom, are "licensed" subject to the following stipulations:

i) The license is made upon opening the package by the customer;

ii) It can be returned to the place of "purchase" (sic!) within 7 days to obtain a refund.
iii) The software may be operated on one computer at a time;
iv) The customer may make one back-up copy which becomes the property of the licensor;
v) Software and data are provided "**as is**" and the licensor excludes all warranties, "except that the media (disk or tape) are free from defect and materials under normal use for a period of 90 days from the date of delivery."

Except for Germany, there seems to be no precedents relating to the legal qualification of "shrink-wrap" licenses or to the validity of typical restrictive covenants found in such standard contracts. Legal commentators agree that some "shrink-wrap" licenses can be found partially or totally unenforceable in the light of general principles of the applicable law of contract (unconscionability). In Germany "shrink-wrap" licenses and disclaimers are strictly controlled by special legislation concerning general conditions of contracts. Such licenses are valid only if they conform to the requirements of the body of case law established under the Statute.[80]

Others correctly observe that certain restrictions may conflict with the principle of **free movement of goods** within the Community or the doctrine of exhaustion of copyright. It should be mentioned that according to a decision of the Supreme Court of Germany, standard programs fixed in tangible media and distributed to the recipient for an indefinite period of use in exchange for a fixed payment are presumed to be classified as "**sales**".[81]

Similar doubts exist in the United Kingdom and France. Restrictions on reverse engineering and decompilations may be viewed as contrary to public policy. The British Copyright Act considers "**research**" as a privileged act, regardless of its purpose.[82] However, the validity of an express prohibition of such activities in the context of a contract of adhesion is an unsettled issue.[83]

Finally, it is worth noting that Section 56 of the British Copyright Act of 1988 indirectly applies to "shrink-wrap" licenses. The Act provides that, if any work in electronic form is marketed on terms that it may be copied or adapted, the right to do so passes to any subsequent transferee of any of such copies.Thus, S. 56 admits express terms prohibiting transfer or making adaptations.

3. Sales of Software and Mixed Software/Hardware Transactions

(a) Problems of Classification

Although the problem of legal classification of software acquisition contracts in EEC countries has not been settled, there is a growing tendency to treat many transactions in **standard software**, as sales or sales-like agreements. In Germany, there are several Supreme Court decisions recognizing software acquisition contracts as sales when the data are incorporated in a tangible medium and the acquirer obtains the right to use only an object code.[84] A similar trend is visible in the United Kingdom. In **Endodynamic Systems**

v.General Automation, the court held that at least contracts involving both software and hardware should be treated as transactions in "goods".[85] Furthermore, the application of the Sale of Goods Act (1979) to software acquisition contracts is rarely questioned.[86]

In Germany, mixed software and hardware acquisition contracts are often classified under the rubric of "sales". If hardware and software cannot be separated without a detriment to the client, he can rescind the whole contract even if only the software is defective. If a defective software can be replaced only by a new version of the program with which the supplied hardware is not compatible, the client can rescind the whole agreement despite of the fact that the tangible element is error-free.[87] The practical difference between such unitary contracts and truly "mixed" contracts consist in that the latter are split into two or more separate parts to which different provisions of the Civil Code are applicable. For instance, a contract for the commissioning of software and acquisition of a suitable hardware equipment was held a mixed agreement governed by the Code provisions relating to contract for work (*Werksvertrag*) and sales (*Kauf*).[88]

(b) Warranties for "Physical" Defects

After a short period of doubt it was accepted that the Code concept of "defect" is applicable to software transactions. Indeed, especially German courts have demonstrated their tremendous capacity to adapt the Code rules on breach of warranty in our field. Section 459 of the Code states that a thing sold shall be free of defects that reduce its value or usefulness for ordinary use or its fitness for the purpose contemplated in the agreement.

The German case law generally applies very strict standards to the seller's obligations relating to the functionality and usefulness of software. Judges found defective performance by the seller of software in the following circumstances:

i) Unclear documentation;[89]
ii) Lack of compatibility between error-free software and hardware recommended by the seller;[90]
iii) Lack of a signal indicating that a diskette is defective;[91]
iv) Lack of proper instructions; manuals shall be translated into German;
v) The use of a program "lock" is permissible but it constitutes misuse if the buyer is not informed about its presence and the seller uses the device to "persuade" the client to buy additional products.[92]

German courts apply rather strict standards to the consulting obligations of the supplier of software, especially with respect to consumers. Furthermore, judges are eager to find the existence of a representation of promised characteristics of a sold program by the seller. The lack of the promised characteristic of the "merchandise" is treated as a "defect" (& 463 of the Code). Such a warranty of "compatibility" can be granted not only by way of express representation, but it can be implied when the seller knew the purpose of the contemplated application of the purchased program.[93]

(c) Remedies for Breach of Warranties and the Impact of General Conditions

Remedies for breach of warranties under German law include the right to rescind the contract and claim only transactional damages (& 276 of the Code). Full damages are available only in the absence of "promised characteristics of a thing" (*zugesicherte Eigenschaften*). From the buyer's perspective, the Code system of liability for defective performance in the framework of sales law has two disadvantages: First, it has a very short statute of limitation period (six months). Second, the buyer does not have a right to demand correction of defects.

The visible improvement of the legal position of the buyer *vis-à-vis* the seller of software products in EEC countries is partially due to a concerted action of large-scale software users and consumers. Apart from general conditions for the delivery of computer hardware and software for government agencies, the associations of users of computer programs from the EEC have elaborated a Model Law for the Acquisition of Computer Equipment.[94] The CECUA Model Contract applies also to mixed transactions involving software. Although a comprehensive evaluation of the Model Contract is beyond the scope of this report, we will briefly characterize its key provisions pertaining to the matters examined in our study.

First, the CECUA Model Contract defines and stresses the importance of the parties' mutual obligations during the installation of an ordered computer system (Cl. 4-11). The supplier is obliged to familiarize himself with the requirements and local conditions at the installation site. Furthermore, he is bound to give the client necessary advice and check his actual needs with respect to the ordered equipment (Cl.4). Second, following the delivery and installation of a system, the supplier shall conduct necessary tests of all delivered hardware and software elements of the package. Copies of such tests shall be made available to the buyer (Cl.13). Third, the supplier warrants that hardware and software are free of design, execution, function, workmanship and material defects and that they conform to the published specifications and contractual terms and conditions (Cl.15). Fourth, the supplier shall deliver all necessary programs and shall guarantee the client access to software improvements during seven years following the delivery of the system. Fifth, The supplier warrants that all elements of the system are mutually compatible and he shall not modify any interfaces without a written permission of the client. Sixth, the supplier is obliged to deliver all necessary documentation and user's instructions (Cl.21).

Sanctions for breach of warranty of title and warranty of defects are stricter than under the Code or typical seller's general conditions for the supply of computer systems. The period of contractual guarantee is *de minimis* 12 months and the client has the right to demand specific performance (repair) or he is authorized to cure the defect with the help of a third party at the risk and cost of the supplier (Cl. 23). The supplier is obliged to defend the client at his costs against third party claims for alleged or actual infringement of intellectual property rights, including infringement of trade secrets and to reimburse all costs incurred by the buyer therewith (Cl. 30).

A commentator stresses the fact that the CECUA Model Contract is rather unpopular among suppliers, but it serves as an educational tool for clients who negotiate computer contracts.[95] Furthermore some suppliers have introduced specific provisions of the Model Contract to their general conditions. Finally, the CECUA Standards are being more and more accepted by courts.[96]

Of course, standard forms and conditions used by suppliers of technology are less friendly to the "buyer". It is worth stressing, however, that in some EEC jurisdictions such contracts of adhesion are controlled by special legislation. In Germany, for instance, the Law on General Conditions of Contracts[97] prohibits a number of clauses deemed to be oppressive to clients and, therefore, they are either null or unenforceable. For instance, a disclaimer of the right to rescind the contract in the event of seller's failure to repair the sold software was held unenforceable.[98] Likewise, OLG Hamm ruled that a clause aimed at establishing a fiction that the delivered software was accepted by the client immediately upon its delivery, was aimed at shortening a mandatory test period for software transactions and it was therefore unenforceable.[99]

4. Custom-made Software and Peculiarities of Contract for Work (Werkvertrag)

The providers of customized software or computer systems are subject to stricter rules than suppliers of standard products. In many respects such transactions are subject to similar legal standards to those applicable to sales and licenses (*e.g.* the notion of "defect", warranties of title, disclaimers, etc.). However, courts have developed some peculiar rules in this field.

In civil law jurisdictions "sales" and "contracts for work" are governed by separate chapters of the applicable code. In Germany, for instance, the basic difference between the two legal regimes consists in that the principal remedy of the employer in the event of a defective program consists of the right to demand repair (& 633 of the Civil Code) rather than in rescission of the contract. The right to rescind the contract (*Wandlung*) arises, in principle, only if the contractor fails to repair the work within a reasonable cure period (& 634). Finally, if the contractor is in delay, the employer (client) may repair the work at the supplier's cost. National laws may provide divergent warranty and statute of limitation periods for "sales" and "contracts for work".

Several judicial decisions in Germany characterize transactions for writing custom computer programs as "**contracts for work**".[100] Also, mixed transactions covering hardware, software and services are classified as "works" if the **dominant** element of the transaction consists in creating of a specified tangible or intangible work (result).[101] A contract for the elaboration of a computer program (software) has been classified by the Supreme Court of Germany as "***Werksvertrag***".[102]

In many European countries copyrightable programs developed within the framework of contracts for work, R&D or consulting agreements belong to the developer (contractor), unless recent modifications of copyright laws introduced the opposite solution. Of course, the parties to a software development agreement may provide otherwise.[103] Because the developer of software usually owns the title to the program, commentators and courts are divided on such issues as to whether the employer (the client) has the right to demand the delivery of the **source code** of the commissioned software. According to a recent decision of the Munich Court of Appeal, a developer of software under an individual contract was obliged to deliver the source code to the client, if the parties had not concluded a maintenance and support agreement.[104] The opinion explained that since the client had to care for maintaining the program, he needed the source code and the developer's duty to deliver it was implied under the circumstances. The decision is consistent with an earlier opinion of the Supreme Court, which had refused to uphold a similar demand of the client's where the developer was obliged to provide maintenance and support services during the contract.[105] The duty to deliver a source code is also stipulated in the Government Special Conditions for the Development of Computer Programs of 21 January 1986.[106]

The legal status of prohibitions against adaptations and self-repair by the client is uncertain. Software developers frequently include such provisions which, while supported in copyright laws and the general law of contract, might be held by courts as practices restraining competition or against public policy.[107] A German district court ruled that the client is authorized to make the necessary adaptations of the acquired program, unless the contract provides otherwise.[108] The permissible scope of adaptations is limited by the purpose of the agreement. The court explained that such interpretation is consistent both with the general principles of interpretation of contracts and & 39(2) of the German Copyright Act, which allows adaptations of a work if it is not contrary to ***bona mores***.

In **Saphena Computing v. Allied Collection Agencies,**[109] a British court indicated that the commissioner of custom software may be authorized to repair the program if it has legally obtained the source code from the defendant. Making available the source code amounts to an implied license to copy it "for the purposes of their business, including repair of improvement of the object code."[110]

Foreign importers of software and computer systems should study the German "BVB Conditions" (1986).[111] They classify contracts for the development and supply of computer systems as contracts for work and are similar to the CECUA Model Law. They provide, *inter alia*, for a minimum 12 month guarantee period, broadly defined right of use of the program and effective remedies in case of breach of warranties. Apart from the Code remedies, the Conditions stipulate for penalties in the event of the contractor's delay.

As a rule, disclaimers in contracts for work are treated in the same way as exculpatory clauses in other software acquisition contracts.

5. The Implications of the EEC Commission Draft Directive on the Legal Protection of Computer Programs

The proposed EEC Directive on the Legal Protection of Computer Programs has generated a flood of controversial opinions, which epitomize the two contrasting policy approaches in this field. At the risk of over simplification, the dispute can be summarized as follows: big computer companies led by United States dominant companies such as IBM and Apple stress the need to grant software producers the maximum protection. They are in favour of cumulative protection of computer programs under copyright and trade secrets laws and are against "reverse engineering". They have formed a lobbying group known as Software Action Group for Europe (SAGE). The second camp, composed of medium-sized and small companies, computer users and academics, has organized the European Committee for Interoperable Systems (ECIS), favours free competition and a weaker protection of software innovations, consistent with the traditional copyright principles of freedom of exploitation of ideas and dissemination of knowledge.

The gist of the ongoing debate focuses on the legality of "reverse engineering" and the freedom of access to **computer interfaces**. The proponents of the pro-competitive approach (ECIS) argue that the Directive must expressly permit research by means of decompilation (disassembling) software in order to allow access to existing interface specifications and algorithms.[112] The proponents of the opposite view opine that reverse engineering is contrary to the Berne Convention and, as a matter of policy, it would encourage "piracy" and "free riding".[113] Indeed, the latter approach, if adopted, would strengthen the dominant position of large computer firms and stifle innovation. This view prevails also among small and many medium-sized firms in the United States.

Members of the ECIS stress that European firms would be at a disadvantage at "home" because "reverse engineering" is permissible both in Japan and in the United States.[114] While in the United States the courts and legal commentators are divided on this issue,[115] Japanese judges and leading commentators take a more moderate and less protective strategy to the protection of software. It is worth noting that Japanese courts are likely not only to legitimize "decompilation", but also to reject the famous Whelan approach, which held that a computer program is protected against substantial copying of the so-called "structure, sequence and organization".[116]

It is expected that a compromise will be reached along the following lines: reverse engineering will be permissible but subject to substantial restrictions. It was recently proposed by the EC that the decompilation can be used to make a compatible program but not to develop a directly **competing product**. The Commission's amended proposal submitted to the European Council pursuant to Art. 149 of the EEC Treaty restricts the application of the "reverse engineering" privilege to those parts of the original program "whose function is to provide for its interconnection with other elements in a system":

"1. Notwithstanding contractual provisions to the contrary, the authorization of the owner of the rights shall not be required where reproduction of the code and translation of its form are indispensable to achieve the creation, maintenance or functioning of an independently created interoperable program provided that the following conditions are met:

a) those acts are performed by the licensee or by another person having a right to use a copy of a program, or on their behalf by a person authorized to do so;

b) the information necessary to achieve interoperability has not previously been published, or made available to the persons referred to in subparagraph a); and

c) these acts are confined to the parts of the original program that are necessary to achieve interoperability with it."[117]

The Commission's amended proposal does not follow the much broadly worded "reverse engineering" exception recommended by the European Parliament, which allowed it also for the purpose of ensuring the **maintenance of the program**.[118] On the other hand, the Commission has followed the Parliament's suggestion that the Directive should expressly allow a legitimate use of a program — without the authorization of the right-holder — to "observe, study or test the functioning of the program in order to determine the ideas, principles and other elements which underlie the program and which are not protected by copyright" (Art. 5 (5)). Furthermore, the proposal incorporates the so-called principle of "**exhaustion**" of a copyright according to which the right to control the distribution of a program shall not be available after its first sale and importation by the right holder or with his consent (Art.4 (c)). The practical significance of the latter provision may be marginal because standard software is usually **licensed** rather than **sold.**

The last text of the EEC proposals contains a reference to programs "**sold**" or "**licensed**" (Art 5). The proposal provides that all acts necessary or incidental to the use of the program purchased under such circumstances shall be permitted. In a nutshell, Art. 5 prohibits contractual restrictions on use and allows translation, adaptation and other alteration where they are necessary for the use of the program by the lawful acquirer in accordance with its intended purpose. By contrast, a **licensee** benefits from the same rule if the contract "does not contain specific provisions dealing with such acts (Art. 5(2))". It remains to be seen whether the Commission and the European Court will tolerate restrictive use limitations clauses in shrink-wrap and other software licenses.

To sum up, the expected EEC Directive on the Legal Protection of Computer Programs will probably legitimize limited "reverse engineering" and improve the position of "purchasers" of computer programs, thus strengthening to some extent the bargaining position of smaller computer companies and users of software both in Europe and elsewhere.

Problems Peculiar to Transnational Transactions

1. General considerations

The last part of this report sketches issues that are peculiar to transnational software acquisitions contracts. It has been prepared in order to highlight typical problems that may arise in **import** transactions concluded by microelectronics industries from the ECLAC region with suppliers from the United States, the EEC or Japan.

International transfer of technology transactions are similar to domestic agreements but they also involve special issues not present in agreements between firms located within the same jurisdiction. Parties to international transactions should take into account differences in their local laws, government controls on the export/import of software, international tax and antitrust aspects of the contemplated deal, choice of law and choice of forum and implications of the distance dividing the parties to the transaction on their mutual rights and obligations.

Typically, since exportation of the majority of software and hardware requires an export license, the parties to a transaction should expressly provide that all formalities and licenses required in the country of exportation shall be arranged by the exporter. Similarly, the importer of software should be responsible to obtain the necessary import licenses in his country. In addition, because many Latin American countries have transfer of technology import regulations, the importer should familiarize the exporter with pertinent rules in the recipient country.

Parties to transnational transactions are advised to define all key technical and legal terms used in their contracts and properly characterize the legal nature thereof. Ideally, the classification of a given acquisition of software contract should fit the applicable categorization in the law of the recipient country and in the law chosen by the parties to govern civil law consequences of the arrangement. Because of the differences among national laws, it is necessary to define even such standard legal concepts as "**exclusive license**". In some countries this term means that the licensor cannot compete with the exclusive licensee in the licensed territory (*e. g.* in the United States). By contrast, in France an exclusive licensor is only obliged not to grant another license within the same field to a third party.

The distance dividing the parties has an impact on the formulation of maintenance and support obligations, as well as on the clauses dealing with the transfer of risk when the goods are in transit. The importers of software or computer systems are advised to try to acquire the products on CIF terms or to allocate delivery costs between the parties. The ideal solution for the importer would be to negotiate the following clause:

"Exporter shall assume all risks of loss or damage to the imported Program (Computer System) while in transit and cover all costs of transportation between the Exporter's factory and the port of destination."[119]

Of course, exporters of technology are reluctant to offer training, support and maintenance services in distant countries, unless they have their local representatives there. Therefore, importers should carefully negotiate a minimum of support and training. The agreement should specify the number of days or weeks provided for technical training, installation and additional support. A relatively cheap form of support consists in providing such services by telephone, telex or fax. Access to the exporter's support center should be available on a round-the-clock basis, or *de minimis*, during the importer's working hours. Consider the following clause:

"During the term of the Agreement, Exporter shall provide Importer with assistance by telephone (fax) regarding the installation, use and maintenance of the Imported Product. Exporter shall reimburse Importer for all costs incurred by it in connection with defective performance of the Product or insufficient explanation of its operation and maintenance in the enclosed Materials."

2. Choice of Law and Choice of Forum

Even experienced lawyers tend to insist that their domestic laws shall govern international transactions entered into with foreign parties.Familiarity with one's own legal rules is certainly an important factor in this context but it is by no means clear that such a choice is the best option for the partisan of his domestic law. If the parties cannot agree which of the two competing systems should govern their relationship, they often choose a "neutral" legal system. Unaware importers from developing countries often agree to choose Swiss law, which allegedly epitomizes the most neutral legal solutions. In reality, however, Swiss law strongly favours the stronger, professional party, especially the exporter of technology. In the absence of choice of law, Swiss conflict-of-laws rules apply the law of the exporter of technology and this choice reflects a deliberate policy of encouragement of suppliers of technology to choose Swiss law and the Swiss forum.[120] The Swiss Law of Obligations seems to be the most liberal codification of the law of contract, which favours freedom of contract, thus permitting the "seller" of technology to exploit his bargaining position.[121] Of course, Swiss law and the Swiss forum are strongly recommended whenever firms from the Regional Network for Microelectronics for Latin America and the Caribbean (REMLAC) region export their computer products abroad.

The foregoing review of the recent developments in the contractual practice in the United States and in the EEC clearly indicates that the European case law is far more even-handed and more sympathetic to the recipient of computer products. Within the EEC, German law is probably the most advantageous to the buyer (licensee) of software products. Thus, the selection of German law is recommended to Latin American importers of software.They should also translate into Spanish (Portuguese) and use during negotiations the various general conditions for the acquisition of software elaborated in the EEC by users of

software. Members of the Regional Network for Microelectronics in the ECLAC Region should consider adaptation of the German Government software procurement general conditions, examined in Chapter II of this report

While negotiating contracts with United States, European and Japanese firms, importers from Latin America and Caribbean countries should consider choosing the laws of such countries as the Netherlands, Sweden or Austria. It is important to mention that the Austrian Statute of Private International Law, unlike its Swiss counterpart, provides that in the absence of choice of law, transfer of technology transactions shall be governed by the law of the country for which territory a license was granted.[122] This law may become applicable to a software import transaction, if the parties agree to arbitrate in Vienna leaving open the question of the law governing their contractual relationship. In such a case the arbitral tribunal shall apply the conflict-of-law rules of the forum.

While acquiring standard software packages in the United States, Japan or Europe, foreigners should carefully check the geographical scope of the "shrink-wrap" license and other terms of the transaction. Frequently, the recipient of such standard computer programs and materials may discover that he acquired the right to use the copyrighted materials only in the country where the transaction was made or that warranty remedies are available only in that country. In such cases, the acquirer (*e.g.* the licensee) should request an express statement from the supplier of the licensed software that would modify the geographical scope of the license and the terms of enforcement of key contractual remedies. Usually, it is enough to obtain the following representation by the seller (licensee) on the back of the invoice:

"The licensor hereby expressly declares that it is aware that the licensee will use the acquired program in [the name of the country of exploitation of the program] *and grants him the right of use, as defined in the enclosed "Terms and Conditions of License", in that country. All warranties and other contractual rights provided in the aforementioned "Terms and Conditions" will be available to the licensee, except that the licensor will not be responsible for providing services concerning* [*e.g.*installation or support, as defined in Sec..]."

Importers of technology from the REMLAC region should consider selecting one of the small arbitral centers such as Vienna, Stockholm, or Rotterdam. They are cheaper than Swiss fora or the Court of the International Chamber of Commerce in Paris. Besides, they are located in countries pursuing strong public policies in favour of the weaker party and freedom of competition. While negotiating dispute resolution problems, importers of technology should consider a compromise solution, which consists in the adoption of the so-called "home-on-home" clause. It requires the party bringing a suit to attack the defendant in his domestic jurisdiction. Naturally, such a clause encourages the parties to attempt to resolve their controversies by amicable settlement. Clauses of this type are enforceable both in Europe and in the United States.[123]

Finally, the parties to international software transactions should remember that certain matters cannot be arbitrated and that some Latin American countries subject import transactions to the exclusive jurisdiction of their domestic transfer of technology transactions and local courts.Obviously, this aspect of the problem is a matter of concern for foreign suppliers of technology.[124]

Notes and References

1. Schachter: Product Acquisition Agreement: A Form Contract With Alternative Clauses: Association of Data Processing Service Organizations, hereinafter "ADAPSO", (1983), at (i)
2. Careful characterization of a given transaction is very important in transnational agreements to avoid its wrong classification under foreign and domestic transfer of technology regulations, tax laws, etc.
3. Davidson: Reverse Engineering and the Development of Compatible and Competitive Products Under United States Law, 5 Santa Clara Computer & High Technology Law Journal, 399,410 (1989).
4. See Warrington Ass. Inc. v. Real Time Eng. Sys., Inc., 522 F. Supp. 367 (N.D. Tex. 1981); BPI Sys., Inc. v. Leith, 5532 F. Supp. 208 (W. D. Tex. 1981). Contra: Videotronics, Inc. v. Bend Eiectronics, 564 F. Supp. 1471 (D. Nev. 1983). In the latter case the court denied trade-secret protection on the grounds that "a property which is subject to protection under federal patent or copyright law cannot also obtain the benefit of protection under either state unfair competition or misappropriation law" Id. at 1476.
5. Harper & Row, Publishers Inc. v. Nation Enters., 471 U.S. 539, 545-546 (1985).
6. See further Soltysinski: Legal Protection for Computer Programs, Public Access to Information and Freedom of Competitive Research and Development Activities, 16 Rutgers Computer and Technology Journal, 447, 467 (1990).
7. Schachter: Program License Agreement With End User. A Form Contract With Alternative Clauses, ADAPSO (1979), Section 7.
8. Id. at 7.
9. A frequently negotiated "temporary use" license clause reads: "Client is authorized to transfer the license and to use the licensed Program on a back-up computer when the designated computer is temporarily inoperable until operable status is restored and processing on the back-up machine is completed". Id. (ADAPSO standard forms).
10. Bingelow: Computer Contracts Negotiating and Drafting Guide, vol.2. form 8.03-1[3] (1989).
11. 531 F.2d 529 (Ct. Cl. 1976).
12. Id., clause 3(b). Bingelow adapted the language of the license agreement between Com/Code Corporation of Washington D.C. and the St. John Shipbuilding & Dry Dock Co. Ltd. of Saint John.

13. The following definitions are recommended by Schachter, *supra*, note 1, at 3-4: "The term 'use' shall include copying any portion of the licensed materials into a computer or transmitting them to a computer for processing of the instructions or statements contained in the licensed program or materials". The term "affiliates" is defined as "any corporation controlling or controlled Licensee or controlled by a corporation which also controls Licensee". For the purpose of the preceding sentence "control shall mean the ownership of more than 50 per cent of the outstanding capital sotck or other equity interest".
14. Sobel, Einhorn: Software Protection and Licensing (in: Technology Licensing 1989, ed. Sobel), at 403.
15. Notice to IBM Customers of February 8, 1983, International Business Machine Corporation, White Plains, NY.
16. See, for instance, the 1976 Copyright Act, as amended, 17 U.S.C. && 107, 109 and 117 (1980).
17. *Supra*, note 10, form 8.03-1, Cl. 14
18. *Supra*, note 1 at 47.
19. Lubrizol Enterprises Inc. v. Richmond Metal Finishers, Inc., 756 F. 2d 1043 (4th Cir. 1985), cert. denied, 475 U.S. 1057 (1986). By allowing to reject the license, the court left the licensee without the right to use the licensed invention. Although the license dealt with a metal coating technology, its rationale applied to all executory licenses and the majority of software licenses belong to this category.
20. P.L. 100-506. 11 U.S.C. & 365(n)(2) (1989).
21. See Kewanee Oil Co. Bicron Corp., 416 U. S. 470, 476 (1974). Recently, the Supreme Court noted again that "the competitive reality of reverse engineering may act as a spur to the inventor, creating an incentive to develop inventions which meet the rigorous requirements of patentability." Bonito Boats, Inc. v. Thunder Craft Boats, Inc. 109 S. Ct. 971, 982 (1989).
22. *Supra*, note 15.
23. Id. A clause drafted by Bingelow goes even a step further: "Licensee shall not decompile, disassemble, or reverse engineer the Licensed Programs or any of them, or attempt to do so." Supra, note 10, FM 8.0-1 cl. 3(d).
24. National Commission on the New Technological Uses of Copyrighted Works ("CONTU"), pp.31-32 (1978).
25. See, for instance, NEC Corp. v. Intel Corp. N. 67, 434 (N.D. Cal. Feb. 6, 1989). The court refused to condemn the disassembling and listing of an Intel microcode for the purpose of analyzing it and making a competitive program. Contra: Apple Computer, Inc. v. Franklin Computer Corp., 714 F. 2d 1240 (3d Cir. 1983).
26. Bingelow *supra*, note 10, Form 8.03-1 (Cl. # 4(a)).
27. 357 A 2d 105 (Del. Ch. 1975).
28. Schachter, *supra*, note 1, at 9.
29. All examples taken from ADAPSO Sample Forms, note 1 supra, at 10-13.

30. A sample contract elaborated under the aegis of ADAPSO defines "updates" as "program logic and documentation changes and improvements to correct known defects and maintain the operational quality of the Licensed Program", while "enhancements" mean "any program, any part thereof or any materials not included in the Licensed Program and Materials at the time of execution of the original License Agreement ... that is developed for the Licensed Program." They usually cover an added funtion to the originally licensed program. Desrosiers, ADAPSO Product Maintenance Agreement, & 1.05 (1983).
31. Disclaimers of warranties are discussed in sub-section e) *infra*.
32. Schachter *supra*, note 1 at 33. A similar comment is made by Bingelow, who is of the opinion that such warranties mean that the licensee is only indirectly liable for failure to have title. Supra, note 10, Form 8.03-1, Cl.5.
33. Schachter, supra note 1, at 35.
34. Id.
35. Durney: The Warranty of Merchantability and Computer Software Contracts: A Square Peg Won't Fit in a Round Hole, 59 Washington Law Rev., 512-515 (1984).
36. Simon: Computer Law Handbook: Software Protection Contracts Litigation Forms, at 64-65 (1990). Compare also Rodau: Computer Software: Does Art. 2 of the UUC Apply? 35 Emory Law Journal 853 (1986).
37. A bill proposed in California to require specific software warranties was withdrawn after opposition of the software industry. See Section of Patent, Trademark and Copyright Law (1987). Special legislation validating "shrink-wrap" licenses was passed by two states. See *infra* Section 3.
38. Schachter, *supra* note 1, at 28, footnote 1.
39. Note that the disclaimer language is in block letters to meet the requirement of the UCC that the exculpatory clause shall be "conspicuous" in the contract.
40. ADAPSO sample forms recommend, for instance, the following clause: "During the period the Licensed Program is under warranty, [Licensor's] sole obligation will be to correct technical errors or failures ... in the Licensed Program of which the Licensee notifies in writing ... This service will be rendered without charge ..., except for the costs incurred by Licensor for machine time, software delivery, medium ... and reasonable travel and per diem maintenance costs ...". Schachter *supra*, note 1, at 31.
41. 389 F. Supp. 45 (D.S.C. 1979).
42. Chretien-Dar: Uniform Commercial Code: Disclaiming the Express Warranty in Computer Contracts, 40 Oklahoma Law Review, 471 et seq. (1987).
43. 708 F. 2d 385 (9th Cir. 1983). The case dealt with a conflict between an express warranty concerning the speed of a line of terminals acquired by plaintiff and a general disclaimer in the contract.
44. But some licenses permit making a back-up copy, modification and merging of the licensed software into another program used by the licensor for its internal purposes.
45. Bingelow, *supra* note 10, Form 8-04(1) Cl.5.
46. Id., Cl. 8.

47. Compare Kemp: Mass Marketed Software: The Legality of the FORM License Agreement, 48 Louisiana Law Rev., at 88 et seq.(1987); Einhorn: The Enforceability of "Tear-Me-Open" License Agreements, 67 Journal of the Patent Office Sofiety, 509 (1985); Stern: Shrink-Wrap Licenses of Mass Marketed Software: Enforceable Contracts or Whistling in the Dark? 11 Rutgers Computer and Technology Law Journal, 51 (1985).
48. 51 L.S.A. && 1961-1966. A similar statute was passed in Illinois but it has been repealed recently.
49. Vault Corp. v. Quaid Software, 847 F.2d 255 (5th Cir. 1988).
50. Id. at 763.
51. See further a review of case law in this field by Soltysinski, *supra* note 6, at 469-470.
52. United States Code, &117 (1989).
53. Such a solution was recommended by CONTU prior to the 1980 revision of the Copyright Act. There seems to be no rational argument to treat differently licensees and lessees of computer program copies and to deprive them the privileges of making back-up copies and adaptations.
54. This typology does not include software services agreements (e.g. batch processing agreements, remote processing service or professional services agreements).
55. Schachter *supra*, note 1, at i.
56. Triangle Underwriters, Inc v. Honeywell, Inc. 457 F. Supp. 765 (EDNY 1978); RRX Indus., Inc., Lab-Con, Inc. 772 F2d 543 (9th Cir. 1985); Neilson Business Equip. Center, Inc. v. Monteleone, 524 A.2d 1172 (Del. 1987) (lease for a turnkey system).
57. Compare Graphic Sales, Inc. v. Sperry Corp., 824 F2d 576 (7th Cir. 1987). It is worth mentioning that the UCC does not regulate a contract for work (*Werksvertrag*). Therefore, the statutory model of contract of sale remains the only "contractus nominatus" that serves the parties to software transactions and judges as a source of legal guidelines how to regulate warranties, disclaimers and other aspects of similar agreements. The situation is different in civil law countries where parties and judges must choose among several types of codified contractual transactions.
58. "Vouching in" provision of the Code provides that, when buyer is sued for breach of warranty for which his seller is responsible, the former may give the latter a notice of litigation and seller will be bound by the result of the litigation if he does not come in to defend the litigation (&2-607[5]). Compare Step-Saver Data Systems, Inc. v. Wyse Technology, Inc., 912 F.2d 643 (3rd Cir. 1990) (vendor computer systems sued its hardware and software suppliers seeking a declaratory judgement that they were liable if certain actions filed by its customers established defects in "products" acquired from the suppliers by plaintiff).
59. Durney, *supra* note 35, at 522.
60. See generally, Chretien-Dar, *supra* note 42, 488-492.
61. 21 UCC Rep. Serv. 1293 (Mass. Dist. Ct. 1977).
62. 564 F. Supp. 160 (E.D. Pa. 1983).
63. 708 F.2d 385 (9th Cir. 1983).

64. See Chretien-Dar, *supra*, note 42 at 498. & 2-725 of the Code provides for a four-year statute of limitations but the term can be shortened to one year.
65. A survey by International Data Corporation found that more than 50 per cent of those interviewed gave 12 month warranties.
66. Daunt: Warranty Drafting Aid, ADAPSO, at 8 (1985). Alternatively, the author recommends a "money back" guarantee: "If the computer program does not perform substantially with the documentation ABC will refund the fees paid for licensing (or the purchase price if the computer program is sold) the computer program". Id. at 9.
67. 15 U.S.C. && 2301-12 (1990).
68. The German government promulgated seven general conditions for different types of contracts for supply of software products to government users. These contractual conditions are binding upon suppliers, unless expressly modified by parties. See generally, Schneider: Softwarenutzungsverträge im Spannungsfeld von Urheber and Kartellrecht, Munich (1989), 59-65.
69. Bundesgerichtshof (BGH) of 6 June 184 (VIII ZR 83/83), NJW 1984, 2938; BGH of 11 February 1971, Betriebsberater (BB) 677 (1971).
70. The decision of the OLG Stuttgart of 3 January 1986, Computer und Recht, 639 (1986).
71. Pagenberg, Geissler: Lizenzverträge, 557 (1989).
72. *Supra*, note 69.
73. OLG Köln, decision of 10 March 1987, CR 12/1989, at 1087.
74. Pagenberg, Geissler, *supra*, note 71, 557.
75. The concept of "defect" (Fehler) will be examined in the next section of the report dealing with sale of software.
76. Goldrain: National Laws affecting Distribution and End-User Agreements, 13 (a paper presented at Conference: Information Technology: Trading with Europe — East and West, Munich, 31 May to 1 June 1990).
77. This creates an irrebuttable presumption that the licensee knew the defects at the time of delivery of the program and it excludes the suppliers liability. Compare Pagenberg, Geissler, supra, note 71, at 519.
78. IPS Model From of License (1987), Cl.16.
79. IBM, Standard Terms and Conditions (1990), Section A, Cl. 8 B. (A form used in the United Kingdom).
80. See the text accompanying note 96, *infra*, OLG Stuttgart of 2 February 1989, CR 685 (1986) The decision conditions enforceability of disclaimers in "shrink-wrap" licenses on their conformity with rules and precedents established under the AGBG.
81. BGH of 4 November 1987, NJW 406 (1988). However, the holding refers to cases when the standard software is conveyed for "*freier Verfügung*", which is translated by a German software expert as "for indefinite optional applications". Goldrain, *supra*, note 76, at 9. But it can also be translated as "free disposition" and not all "shrink-wrap" licenses grant such broad permissions to the customer.

82. Chalton: Implementing the EC Council Directive: The National Perspective form the UK, (in: Information Technology: Trading with Europe, Conference of International Federation of Computer Law Association, Munich, 31May to 1 June 1990), at 12. Similar view dominates in the French jurisprudence. Michau: The French Perspective of the EEC Directive, 5 (the Munich Conference, *supra*).
83. See further discussion on the issue of "reverse engineering" under the future EEC Council Directive. Section 5, *infra*.
84. BGH of May 2, 1985, GRUR, 1055 (1985); BGH of 4 November 1987, supra, note 81. See further Hoeren: Der Softwareüberlassungsvertag als Sachkauf, CR 908 (1988).
85. As reported in Applied Computer and Communications Law, v. 5. at 58 (1989).
86. Compare Goode: Commercial Law, 154 (1985).
87. Landesgericht Berlin, decision of 2 February 1987, IuR, 424 (1987).
88. LG Augsburg of 11 November 1985: IuR 166 (1986).
89. KG Berlin of 24 November 1985, CR 643 (1986).
90. OLG Celle of 26 February 1986, CR 303 (1988).
91. OLG Köln of 22 June 21988, NJW 2477 (1988). The same holding applies when a program defect is signalled but without explaining the cause thereof.
92. LG Stuttgart of 3 January 1986, CR 639 (1986).
93. OLG Saarbrücken of 30 May 1990, CR 713 (1990).
94. The model contract was published in Germany and is generally known as "CECUA – Standard Contract" (Modellvertrag für den Kauf von Computeranlagen und Geräten).
95. Schneider, *supra*, note 68, at 64.
96. See, for instance, the decision of LG Düsseldorf of 1987, CR 292 (1987), in which the court held that the supplier is obliged to get acquainted with the production requirements of the customer when writing an individual program.
97. Gesetz über Allgemeine Geschäfts Bedingungen (AGBG).
98. LG München of 23 January 1985, (IuR 72 (1986)).
99. Decision of 12 December 1988, (NJW 1041 (1989).
100. BGH of 11 February 1971, BB 677 (1971); OLG Oldenburg of 12 February 1986, CR 552 (1986).
101. See BGH of 24 July 1986, CR 799 (1986).
102. BGH of 30 January 1986, CR 377 (1986).
103. But even in France, it is advisable for the developer to include an express provision that he retains ownership in the program. An example of such a stipulation reads: "All instructions, procedures and computer programs first made by Contractor in the course of developing the software to be furnished to the company remain the property of the Contractor. The Contractor may use the entirety of information and knowledge which he may acquire in the course of development of the software." Muenchinger: Who Owns Software Developed Under Contract? The French Perspective, EIPR 311 (1986). In BMW v. Pachot, a fired ex-employee prevailed in a suit against his former employer over the issue of the ownership of a program developed by the former

without any assistance from his employer but with the use of processing cards removed temporarily from his work place. Cour de Cassation, March 7, 1986, as reported by Muenchinger. Id. at 307.

104. LG München of 18 November 1988, NJW 2625 (1989).
105. BGH of 30 January 1986, NJW 1259 (1987).
106. Besondere Vertragsbedingungen für das Erstellen von DV-Programmen, as published in Heussen: Computerrecht Handbuch, 1 (1990). hereinafter referred to as "BVB Conditions".
107. Goldrain, supra, note 76, at 6.
108. LG Munich of 17 February 1987, CR 379 (1988).
109. Official Referee's Court of 25 April 1988, 59 Computers and Law 20 (1988).
110. Id.
111. *Supra*, note 106.
112. Cornish, Interoperable Systems and Copyright, EIPR 391 (1989); Colombe, Meyer: Interoperability Still Threatened by EC Software Directive: A Status Report, EIPR 325 (1990).
113. Lake et al.: Seeking Compatibility or Avoiding Development Costs? EIPR 43 (1989): Burkill: Reverse Compilation of Computer Programs and its Permissibility Under the Berne Convention, Computer Law and Practice 114 (1990).
114. This view is essentially true although the matter is controversial in those tow countries, too. See Soltysinski, *supra*, note 6, 468-469; Durney: Reverse Engineering Under Japanese Law, Intellectual Property, Marketing and Community Law, 2 (1990). The author cites Japanese authorities for the proposition that under the Japanese Copyright Act "reverse engineering" is permissible. Furthermore, he explains in detail that a recent decision in Microsoft Corp. v. Shuuwa System Trading KK of 30 January 1987, which is interpreted by some partisans of the SAGE camp as allegedly outlawing "disassembling" of software, did not even discuss that issue. Id. at 3. See further, *infra*, note 115.
115. Contrast Whelan Inc. v. Jaslow Dental Laboratory Inc. 797 F2d 831 (1987) and Apple Computer Inc. v. Franklin Computer Corp. 714 F2d 1240 (3d cir. 1983) with NEC Corp. v. Intel Corp. 10 USPQ 2d (ND Cal. 1989). The latter decision found "reverse engineering" unobjectionable.
116. Compare System Science Corp. v. Toyo Sokki KK, Tokyo Dist. Ct decision of 31 March 1989, commented by Karjala: Japanese Courts Interpret the Algorithm Limitation on the Copyright Protection of Computer Programs, 7 European Intellectual Property Rev. 235 (1990). See also Karjala: The First Case on Operating Systems and Reverse Engineering in Japan, 10 EIPR 172 (1988).
117. Commission of the European Communities, Amended Proposal for a Council Directive on the Legal Protection of Computer Programs of October 18, 1990, COM (90) 509, at p. 28.
118. See Official Journal of the European Communities, No. C 231/78 of 17 September 1990, Art. 5 A.

119. Naturally, exporters of software systems propagate the opposite solution. See Palenski: Exclusive Distribution Agreement (International), 6 (1983) (ADAPSO FORMS).
120. Compare Loi fédérale sur le droit international privé of 18 December 1987, Art. 122.
121. See further Soltysinski: Choice of Law and Choice of Forum in Transfer of Technology Transactions, Recueil des cours, 307-323, 345-347.
122. See the text of the Austrian Statute on Private International Law published in: 43 Rabels Zeitschrift 383 (1979), & 43.
123. See Smith, Valentino & Smith, Inc. v. Superior Court., 17 Cal. 3d 491 (1976).
124. See generally, Correa: Transfer of Technology in Latin America: A Decade of Control, 15 Journal of World Trade Law, 388 (1981); Soltysinski, *supra*, note 111, at 249 et seq. Recently, however, many Latin American countries have liberalized their transfer of technology laws.

The Legal Protection of Software*

C. M. Correa**

I. Introduction

This paper discusses, from an economic perspective, the main issues involved in the legal protection of computer programs, particularly as they concern Newly Industrialising and Middle-Income Economies (NIEs and MIEs).

Section II briefly analyses the characteristics of the world software market and production in order to set out the context in which the protection issue is dealt with. It holds that in this area there is a profound North-South technological and industrial asymmetry and that the prospects of developing countries to enter into this field are more limited than often claimed.

Section III presents the main legislative trends regarding software protection and the rationale underlying the prevailing copyright approach. It also examines the ambiguities and uncertainty created by the application of copyright law in this area, and the growing dissatisfaction with its coverage and effects.

The implications of software protection for the diffusion and local production of software are discussed in Section IV. While the granting of some form of protection seems necessary for political or economic reasons, it is argued that its effects on the access to computer programs and on their development depend on the structure of the market and the country's relevant policies.

On the premise that no universally valid form of protection is sustainable, Section V finally addresses some of the regulatory aspects that may influence the diffusion of productive software policies in NIEs and MIEs. It suggests that there is no general prescription on how to formulate an adequate legal strategy on the matter, and that the form and extent of software protection should take into account the economic and technological conditions as well as the objectives of the concerned countries.

* Research for this report has been funded by the OECD Development Centre as part of its research project on "Technological Change and the Electronics Sector — Perspectives and Policy Options for Newly Industrialising Economies". The author is grateful to Dr. Dieter Ernst for his stimulating comments and suggestions. He would also like to thank participants of the OECD workshop on the electronics industry (Paris, June 1989) for their comments on an earlier draft of this paper.

** Doctor of Law, Coordinator of Project DP/RLA/92/014, Buenos Aires, Argentina

The main conclusions are presented in Section VI.

II. World Software Demand and Supply

1. World market: main features

Software nowadays constitutes one of the most dynamic segments of the information technology market. In 1979 the world software market accounted for an estimated US$48 billion; it grew at nearly 22 per cent annually in the period 1984-1987 (OECD, p.21, 1988).[1] OECD countries accounted for nearly 97 per cent of the world market in 1984. The United States domestic market represented 54 per cent thereof. Only a few developing countries rank according to an OECD study (OECD, 1985) among the major software markets in the world. Brazil would be the tenth major national market,[2] quite far from Mexico (12th) and the Republic of Korea. The United States accounts for a major part (around 70 per cent) of world software production, followed by France and Japan (United States Department of Commerce, 1984). Analyses at the country level indicate for most countries, including developed ones, that a significant part of the market consists of imported software distributed by local dealers or by subsidiaries of foreign enterprises. This applies particularly to basic software and various types of standard application software. Custom application programs, instead, are *de facto* reserved to a great extent to local firms.

The United States software industry is the most internationalised one among those of OECD countries. A significant part of its worldwide revenues have a foreign origin. France ranks second according to the level of internationalisation of its industry (mainly based on the provision of custom software); United Kingdom, Canada and New Zealand follow. Japan presents one of the lowest levels of internationalisation within OECD (OECD, 1988, Table 24).

Although no specific information is available, it is safe to affirm that the world market share of developing countries is in a 3 to 5 per cent range and that it almost entirely corresponds to application software for domestic markets. Some NIEs have initiated attempts to develop an export-oriented software industry. However, their results are still marginal in global terms.

2. Do the NIEs and MIEs enjoy any competitive advantages in software production?

The determinants of competitiveness in software markets have not been thoroughly studied yet. The dimension of the domestic market and the size and marketing capabilities of the United States' firms may explain their success at the national and international level (OECD, p 51, 1988). In most other countries, including France, the limited size of the market seems to be a significant restriction on the growth of the software industry, particularly on expanding towards standard software (Correa, 1987). In the case of Japan (the second largest country by the number of computers installed), the emphasis traditionally put on custom software and the barriers imposed by language may be some of the

factors that explain a very low degree of participation in the international market, notwithstanding the size of the domestic market and the fact that Japanese programmers are reported to be many times more productive than their American colleagues (United States Department of Commerce, p. 11, 1984).

In many developing countries software production has been identified as a promising field of action. It is argued, newcomers face high barriers for joining the production of hardware, while with low capital investment and the mobilisation of local qualified personnel, it is relatively easy to exploit the growth potential of the software sector. Paradoxically, a few NIEs have evidenced an ability to break into some segments of hardware production (e.g. microcomputers and peripherals) successfully, while the efforts made to establish software capabilities have not had, at least up to now, significant results.

A number of factors may favour the development of software in developing countries. Among them, low wage scales for computer professionals seem the most clear cut advantage. In countries such as India, Brazil and Argentina, local salaries may be many times lower than those prevailing in OECD countries (Katz, 1986; Takahashi and Pereira Lucena, 1988; Subsecretaria de Informatica y Desarrollo 1987). There may also be advantages stemming from external circumstances, such as growing software backlogs and scarcity of personnel in developed countries, the proliferation of international subcontracting, etc. (see Table 1).

At the same time, however, there are a number of facts that considerably dilute the real possibilities for developing countries to break into the software field.

Factors favoring the development of software	Factors retarding the development of software
Low wage scales	Small domestic markets
Growing software backlogs	Low capital availability
Increasing development, operating and maintenance costs	Lack of market expertise
Lack of specialized software for local conditions	Absence of an informatics or computer industry policy
Proliferation of International subcontracting for software development; joint training centers	Absence of taxation/fiscal and R&D incentives for software producers; regulatory restrictions on importation of technology and software
Local support services requirements Modifications requested by users	Shortage of labor with required skills; retention of highly skilled labor necessary
New communications technology	Shift toward semi-automated programming Language barriers Severe competition from large companies in R&D and marketing Difficulties in providing adequate maintenance and support

Table 1. Factors in the Development of Software by Developing Nations
Source: Schware, 1987

In addition to the smallness of domestic markets (an aspect which plays a part even with countries like Brazil), there is generally a shortage of professionals actually qualified to develop software in accordance with international standards, as well as the management of software development projects of a certain complexity.[3] Moreover, even if those skills are available, the marketing of software, and particularly the access to extremely competitive markets such as the United States one, poses extremely difficult problems (Katz, 1987). It is not enough to develop a good software; it is necessary to know how to sell it.

A survey made in Argentina with major local software producers revealed that most firms considered that their comparative advantages (availability of qualified personnel, low salaries) were not sufficient to compensate the obstacles for software development and commercialisation. The obstacles more often cited included the small size of the market, the lack of resources and capabilities in R&D and in marketing, and limitations as to capital investments.[4] In connection with the export of software, the difficulties concerning marketing and distribution and the post-sale client support were particularly mentioned (SPCA-LAI, 1988). Moreover, the mere identification of a concrete potential demand is problematic, when there is no proximity with the potential user. For this reason, the establishment of subsidiaries for joint ventures may be an essential instrument to enter foreign markets in this field (Correa, p.8, 1987).

III. Legal Protection

1. Main legislative trends

The issue of legal protection of computer software appeared when software affirmed itself as a good that could be traded separately from hardware, and particularly with the expansion of "packaged"" software. Before 1983, only three countries had specifically legislated on the matter: Philippines, United States and Bulgaria. After that year more than a dozen countries introduced rules regarding software protection: Hungary (1983), Australia (1984), Federal Republic of Germany (1985), France (1985), India (1985), Japan (1985), United Kingdom (1985), Taiwan Province of China (1985), Republic of Korea (1986), Spain (1987), Singapore (1987), Malaysia (1987), Indonesia (1987), Brazil (1987) and Canada (1988).

The determination of the appropriate legal framework for the protection of software gave rise to considerable debate in both developed and developing countries. In some of them, attempts were made to devise special rules for software protection, in order to take into account its functional character and the peculiarities of its commercialization and use. In Japan, the Ministry for International Trade and Industry (MITI) proposed a special regime in 1983 in order to exclude moral rights, limit protection to 15 years and regulate the use of software on terms balancing the private and public interest. In France, the National Institute of Industrial Property also proposed a *sui generis* optional protection (1984). In Brazil and

Argentina also some draft laws proposed special rules (though, in the latter country, having copyright as the general framework). Most of these proposals have been abandoned by now (see also point 2 below).

The protection of software under copyright laws is the predominant trend worldwide. Apart from cases where specific amendments were introduced to such laws, in a number of other countries judicial or administrative decisions also followed that direction (Switzerland, Belgium, Italy, Mexico, Chile, etc.).

In most cases, the adoption of the copyright approach has been instrumented by amendments to copyright laws which specify that software is a copyrightable work as are the rights relating to copies and adaptations. In a few countries the reforms have been deeper, such as Japan and France (Correa et al., 1987, p. 116) as well as in the Republic of Korea, Brazil and Indonesia.

All developing countries that have already adopted legislation in order to legally cover computer programs have admitted the copyright principles. The threat of the application of Section 301 of the 1984 United States Trade Act has prompted some countries to deal with the issue in accordance with that approach.[5] In Brazil, the "Software law" of 1987 regulated the application of copyright to computer programs, but also created a detailed regime for the commercialization of such programs in the country.

2. Rationale for copyright protection

Abundant literature has analysed the different legal institutes under which software may be protected, namely copyright, trade secrets, contractual law, patents and a special regime. The application of utility models has also been proposed (Higashima, p.12, 1986). As mentioned before, the prevailing trend, after some unsuccessful attempts to establish special regimes, is software protection under copyright.[6]

The referred trend has been strongly influenced by the United States position on the subject, particularly after the amendment, in 1980, of the United States copyright law. In turn, the option for this form of protection has been determined to a great extent by the domestic and international interests of large software producers. the main advantages for them in relying on copyright derive from:

- the possibility to apply well-known and generally respected principles and rules;
- the assimilation of software producers' rights to those of literary, artistic or scientific authors, in spite of the functional character of programs;
- the access of established legal remedies against unauthorized reproduction;
- the long-term of protection conferred;
- the commencement of protection since the date of the creation of a program;
- the lack of registration requirements to obtain protection; and

- the existence of international conventions where protection is obtainable on a universal basis.

The last point mentioned is crucial for the international operation of the industry. to the extent that the copyright approach is admitted, under the Universal or the Berne conventions, a computer program created in one country automatically receives protection in almost any country in the world.[7] The monopoly rights granted facilitate commercial exploitation of such programs on a worldwide basis. The stronger the protection, the less is the need to be present (through a subsidiary or licencee) in a particular market (Correa, 1988b). The world market can thus be supplied under the highly centralized productive scheme that prevails in the software industry, at least wherever standard products can meet the users' demands and there are no other compelling factors for some form of permanent establishments.

Conversely, copyright offers some disadvantages from the producers' standpoint. The main one is that it is conceived to prevent copying and not the use of a protected work. Henceforth, the legal power to prevent unauthorized use (including private use) is limited. Another problem may arise in connection with the originality requirement. In some countries where high standards are applied (as in Germany), many computer programs may not qualify for protection.[8] In fact, in many cases a piece of software is determined by functional specifications in such a way that the scope for originality is very restricted or non-existent. In addition, copyright only protects the expression of a work, but not the underlying idea. It therefore allows third parties to base any new development on an existing idea, even if the latter's expression is protected.[9]

On the other side, the impact that the introduction of protection may have in fostering a domestic industry is quite uncertain. Protection is particularly important for standard software, and especially for packages that run on microcomputers. Unauthorized copying of bigger systems is more difficult given the suppliers' proximity (through maintenance and other services) to equipment installations. For custom software — which is precisely the area in which domestic firms mostly work in NIEs and MIEs — contractual provisions may be far more important for protection than any general legal regime.[10]

From the point of view of the user, copyright exhibits many disadvantages, which come from the original conception of that legal system. Designed to protect intellectual works as an emanation of human creativity, it is strongly biased in favour of author's rights. While many rights accrue to him, obligations are minimal. Unlike patents, for instance, no working obligation is generally established. At the same time, protection may be obtained even without disclosure of the work. the long terms of protection (generally fifty years *post mortem auctoris*) do not allow the society to benefit from the free use of the work (in this case a technical functional work) within a reasonable period after its development. Furthermore, as stressed by MITI's proposal of a special regime, that system does not contain

provisions to guarantee the user against defects or lack of support for the use of programs (MITI, 1983). Finally, the granting of "moral rights" contradicts the nature of software as a living entity, which is constantly adapted and improved.

3. Copyright Questioned

In the light of the difficulties of treating software as a copyrightable work and of the shortcomings referred to, it is not surprising to find criticism and several reservations on the copyright approach, even in developed countries where it has been formally adopted.

Dissatisfaction comes from many sides. Producers are unhappy with the limited effect of copyright on actual copying. Producers' associations claim continuous losses due to piracy in the United States and other countries. Surveys made in the United Kingdom and the Netherlands, for instance, indicate a general lack of confidence in the protection provided for computer programs by copyright law. Only 15 per cent of the respondents (in the case of the Netherlands) stated that they were prepared to enforce their legal rights in civil courts in case they were confronted with software piracy. This attitude results from the lack of a clear, unambiguous legislation (Borking, 1987). On their side, users are often confronted with too restrictive clauses, for example, in connection with archival back-up copies (Meisner, p.397, 1988) and educational purposes (OTA p. 8, 1986). For instance, a highly controversial draft bill was introduced in April 1988 in France in order to allow universities and graduate schools "to reproduce the software they have acquired for their educational activities, provided that these copies are not used outside of those universities and schools" (Bertrand and Coust, 1988).

In the United States, the policy on software protection states a study of the Office of Technology Assessment, "is being made in the courts, virtually on a case-by-case basis, and the resulting ambiguities satisfy no one" (OTA, p.34, 1986).

Case law has, in effect a decisive role in shaping the scope of protection afforded in that country. One major development has led to a re-interpretation of the principle that confines copyright protection to the program's expression. In *Whelan Associates vs. Jaslow Dental Laboratory*, while recognizing that copyright protection does not extend to the "idea" or functionality of the program, the court held that it covers the sequence, organization and structure of the code-program.[11] Furthermore, in *Broderbund Software vs. Unison World* it was decided that the protection of the underlying program extends to all elements of its audiovisual display.[12] Courts also face the need to decide on the imprecise frontiers of copyright protection in specific cases. After an intense debate they decided to support the copyrightability of "microcode" — which controls the sequence of operations carried out within the computer in response to a particular instruction received — in *NEC Corp. vs. Intel Corp.* (Sandison, 1987) despite its clear mechanical and utilitarian nature.[13]

In *Alloy vs. Ultratek*, moreover, the copyrightability of hardware itself in the form of Programmable Array Logic chips (PALs) is at stake. If the decision is affirmative, "then hardware — at least its low-level, step-by-step functionality — would quality as a 'work of authorship', placing virtually all unpatented logic devices (generally presumed to lie in the public domain) under the protection of copyright law" (Siegel and Laurie, 1989).

In other pending cases (based on suits by Lotus Development Corp., Ashton Tate Inc. and Apple Computer Inc.) judges are bound to decide whether a software company can legally protect a program's appearance, design and functionality — its "look and feel". If granted, such a protection would include visual program features as pull-down menus, graphic symbols and even certain key stroke sequences. This eventual further extension of copyright has already brought up considerable criticism, and raised questions on the capabilities of United States software firms to compete on the basis of innovative ideas rather than on the basis of legal instruments (Burgess, 1989; Business Week, Editorial, p. 22, 1989).

The confusion on the means to ensure the legal protection of software has increased recently in the United States due to the so far successful attempts to ensure patent protection for computer programs. Recent evidence indicates " that all software claims are eligible for patent protection unless they simply involve the use of a mathematical formula to calculate and display a number. Software patentability is a de facto reality today, as the Patent and Trademark Office (PTO) now commonly issues patents for software inventions" (Maier, p. 157, 1987).[14]

The inadequacy of copyright protection should, in view of the United States Congress Office of Technology Assessment (OTA), lead to the development of a new legal framework:

> *"The distinction between writings and inventions is indeed breaking down with respect to functional works such as computer software and semiconductor chip masks. Because there are many works of this type, they may require their own framework for protection. If it were based on the distinctive characteristics of these works, the law might be more accurately targeted to achieve specific policy outcomes, thus serving a more robust policy tool. With a new category of law, both producers and users would face less uncertainty each time a new type of work were introduced. OTA 's analysis suggests, too, that a fruitful basis for a revision along these lines might be found in the distinctions between works of art, works of fact, and works of function" (OTA, p. 14, 1986).*

Paradoxically, OTA recommends an approach that, as indicated before, the United States Government has strongly opposed, particularly in Japan. The need to look for a special form of protection was also stressed in other countries when amendments to their respective copyright laws were proposed or approved. In France, the rapporteur senator Jolibois qualified software as being of "industrial character". Moreover, it was stated that the law was "approved as a temporary measure, still remaining as an ultimate objective the search for a specific form of protection which will surely require some years to be found" (*Journal*

Officiel, 1985). In Australia, the Minister of Justice referred to the 1984 amendment in his country's legislation as "a solution for the short term", which should allow to completely revise the policy adopted for the long term. In Canada, the study "From Gutenberg to Telidon — a white paper on copyright" published in 1984[15] understood — like some judicial decisions in several countries — that the object program was not protectable under copyright law. A special title for ten years was proposed.[16]

It should also be recalled that the specialized United Nations organisation on intellectual property, the World Intellectual Property Organisation (WIPO), proposed in 1978 a set of model specific rules on software, later on abandoned as the copyright approach became prevalent. The WIPO's recommendations have been the basis, however, of many initiatives such as the comprehensive computer draft law recently distributed by the Ministry of Justice of Israel (Levenfeld, p. 5, 1988).

Many authorities have objected to or made reservations on the application of copyright to software. Trolle (Switzerland) advocates that software is an intellectual method, not a creation. It would lack esthetic character (Ulmer and Kole, 1983). Desjeux (France) stresses that intellectual property is an "hommage" of society to "creators" (moral rights, long term of protection, etc.). The inventor receives more limited rights, like the software producers should, since the latter make an "intellectual contribution" but do not "create" (Desjeux, 1986). Van der Berghe (Belgium) argues that the lack of human communication in software conspires against the fundamentals of intellectual property (Flamee, 1985). G. Shipley (United Kingdom) affirms that software is different from protectable works both for its origin and use (Shipley 1985).[17] Jean Jonqueres, Presiding Judge of the Supreme Court, in Paris, after analysing the disappointment with software protection through copyright, concludes that the protection by a patent is likely to be even more disappointing "in view of the traditional strictness in applying the criteria of patentability and the interpretation of the claims. In the absence of any general text governing the protection of intellectual property, would it not be better to move towards a protection *sui generis*? This, with the protection provided by legal proceedings for unfair competition, is the only satisfactory protection for intellectual creations" (Jonqueres, p. 620, 1987).

Briefly, copyright has not yet succeeded in becoming an uncontested and satisfactory framework for software protection. It is likely, in fact, that even if it is admitted that software deserves a legal protection, the debate over the form that it should assume will continue in the future. A crucial point is how a proper balance among the different interests at stake can be reached.[18] Of course, such a debate is of utmost relevance for developing countries, particularly for those which intend to formulate active policies with regard to the diffusion or local production of software.

IV. Implications for Software Diffusion and Production in NIEs and MIEs

The analysis made in the precedent sections indicates, first, the existence of a profound North-South asymmetry in technological and productive capabilities for software development; second, that notwithstanding some efforts, the NIEs and MIEs have not been able to achieve significant positions in the software field; third, that the existence of given comparative advantages for software development in those countries is questionable.

On the other side, Section III has shown that considerable uncertainty and ambiguity prevails in connection with the extent of protection conferred by copyright.

What implications may the prevailing software protection patterns have on NIEs and MIEs in this context? This question should be dealt with in relation to two aspects: the diffusion and the local production of software.

From the point of view of diffusion, liberal copying would arguably reduce the cost of access to software. In the last analysis, suggests Prof. Wells, for a country which is not an innovator in the field it may be convenient, from an economic perspective, to facilitate the obtaining copies at low costs to stimulate a rapid software diffusion and save foreign currency (Wells, 1987). High software prices[19] may make it difficult for domestic firms to computerize and compete internationally. Important trade-offs may exist, however, whether protection is granted or not. The lack of appropriate maintenance and after-sales support, and the consequences thereof for an efficient application of computer programs, may limit the advantages of non-protection. On the other side, while licensing under copyright may slow the diffusion of certain types of software, it may at the same time support the introduction into the economy of high-quality types of software. From an international point of view, moreover, a free-copying approach would be extremely conflictive. It does not seem feasible nowadays for a country to compete by departing from generally accepted rules in intellectual work protection.

The initiatives for strengthening and internationally expanding the legal protection of software have almost completely disregarded the problems posed for developing countries. The establishment of some form of protection will, in the first place, work in favour of those enterprises already operating in the market. It will eventually reduce piracy and increase the income obtained through the distribution of a larger number of copies, at a higher price. Firms exporting software to the protected market would be among the main beneficiaries of the legal change. It is noticeable, however, that according to an OECD survey, the lack of protection by national law is not deemed by exporting firms to be a "high" obstacle for international operations, but just one of "medium" importance (OECD, p. 65, 1988).

Again, the impact of protection considerably differs according to the type of software developed. It may eventually have a significant impact if national firms intend to compete in the area of packages; this is, however, a considerably limited possibility due to the size of local markets, the investments needed and the difficulties in specifying standardized products for distant potential users. If software development basically means production of custom programs, legal protection will not add very much to the existing situation.

The surveys made in some countries illustrate the software suppliers' point of view on the issue. The information collected in Argentina and the Republic of Korea revealed a general attitude in favour of legal protection[20]. In both cases, however, an important proportion of respondents indicated their preference for a special regime rather than for copyright (90 per cent in Argentina; 42 per cent in the Republic of Korea). Moreover, in the case of the Republic of Korea, the majority (97 per cent) "feared that the implementation of such protection at too early a date would hamper the growth of the domestic information industry." (Song, p. 5, 1987)

In sum, to the extent that a local industry is confined to or concentrates itself on custom programs, the effects of legal protection will mainly reflect on imported software. Even in the case where packages are also produced, it cannot be assumed — obviously — that the introduction of protection or of a strengthened regime will lead automatically to more and better local production. The legal framework will be one factor that may influence the software development, but in no way may it be deemed to be the most important or even a significant promotional element. The protection conferred may eliminate the unfair competition of pirated programs sold for a few dollars. This positive effect may be counterbalanced, however, by a stronger competitive position ensured to importing firms and, eventually, by a larger presence of foreign companies in the local market.

Another aspect to be considered is the situation of a country that does not confer protection and is willing to export software to third countries. Under present international conventions (Berne and Universal) the Member countries are only bound to grant foreigners "national treatment". This rule would not be violated if neither foreigners nor nationals were granted protection. It is doubtful whether it can be interpreted that those conventions cover computer software within their widely defined scope. However, present initiatives of the United States at GATT precisely aim, among other things, to establish software protection under copyright as a universal standard. Japan and the EEC also share this proposal, notwithstanding some differences as to the content of the standards and norms to be developed (Correa, 1988b).

In any case, it seems clear that the development of a local software industry will not necessarily be benefited — it may also be jeopardised — by the establishment or strengthening of a legal system of protection. The promotion of a software industry will require more complex and specific instruments than simple protection. The experience of many countries — Brazil, India, Republic of Korea — indicates that special policies have to be implemented with that aim (see point II.3 above).

V. Options for NIEs and MIEs: Key Issues in Devising Legal Regimes for Software Protection

The newness and complexity of the protection issue, and the confusion existing in developed countries, make it extremely difficult for a developing country to adopt decisions on the matter. As mentioned before, dissatisfaction with the copyright approach is important and growing. The patent system does not seem to offer a better solution. It makes protection stronger since even independent developments on the basis of the underlying ideas of a program would be excluded. The setting up of a special regime, finally, faces the difficulties inherent to the creation of a completely new legal framework, particularly *vis-à-vis* its recognition in other countries.

Independently of the approach followed, a number of key issues need to be considered if certain industrial or diffusion objectives are sought.

1. Subject matter and scope of protection

While recognising that protection extends to a computer program in its source or code form, or even embedded in a Read Only Memory (ROM), the development of the industry requires that the ideas themselves do not become directly or indirectly the property of the program title holder[21]. In this sense, the Japanese law explicitly excludes from protection the algorithms and rules employed in the development of a program. Likewise, languages should not be considered copyrightable. Only the expression of a program is to be deemed protectable, if some room for alternative creation of software is to be retained.

2. Duration

The typical duration for copyrights, as mentioned before, generally extends beyond the author's death. In the case of works of juridical "persons", periods of 50-70 years are the rule. These terms are clearly incompatible with the diffusion of computer programs while they are still economically and technologically valuable. Moreover, the recovery of investments made in the development of a program is often completed in a few years. The extension of the exclusivity would only ensure a monopoly rent for the title holder and prices for users higher than those obtainable under free competition. While adopting the copyright framework, some countries (France, Brazil, Indonesia) have limited its duration to 25 years for computer programs.

3. Adaptations

A crucial point for countries which are strongly dependent upon imported software is to allow some flexibility for adaptation of programs, either to specific types of equipment (this would be particularly important if a local hardware industry is promoted), or to local conditions. The Brazilian law, for instance, stipulates that when provided for in the contract, the rights on the technological changes and adaptations will belong to the person authorized to make them, who will exercise those rights autonomously (Art. 6, Law 7646).

4. Copies

Developed countries' laws tend to restrict the right to make copies[22]. Three main regulatory lines seem to exist:

1. Copies are permitted by law, under specific conditions (United States, France, Japan);
2. Copies need always to be authorized by the proprietor (Germany, United Kingdom);
3. Back-up copies are permitted by law, except if prohibited by the proprietor (Australia) (Correa, 1988a).

A broader right to make copies may be necessary, however, to reach a balance between the title holder and the user's interests. The diffusion of software may, in particular, be hindered by too stringent provisions on this aspect. The Brazilian law permits the legitimate user to make all copies "indispensable for adequate use" of the program (Art. 7, Law 7546). The Republic of Korea 1987 law, for its part, allows reproductions for use "for the individual purpose in a limited place like home" and for educational purposes, among others (Art. 12, Law No 3920, Dec. 31, 1986)[23].

5. User's points

Another important regulatory aspect relates to the rights for the continuous use of a program. Since, under copyright, registration is neither compulsory nor ensures full disclosure, in certain situations — such as when the title holder has gone out of business or cannot be contacted — the user may be in a very difficult position. The Republic of Korea law, in a quite original provision, stipulates that if the owner of the program copyright is unknown and cannot be located, the user may apply to the Ministry of Science and Technology for approval to use the work. In such cases, a deposit of compensation for use of the program will have to be made with the Ministry (Art. 17). In order to facilitate the access to computer programs Article 18 of that law provides, further, that a program copyright holder must allow a *bona fide* user to use a program which has already been published and distributed unless there is justification for not doing so (Art. 18)[24].

For its part, the Japanese law does not deem the use of a program for non-commercial purposes to be a copyright violation when the user does not know about the infringing character of the copy.

As the preceding discussion reveals, the regulation of software protection may — even within the framework of copyrighting principles — reflect certain policy objectives related to the diffusion or production of programs. How to obtain a balance between the private and public interests, including those of users as well as of local industry, is the crucial point for the formulation of strategies on software protection.

It should be clear, in particular, that no general prescription on the matter can be made. There is nothing in the nature of software as an economic and technological entity that would justify a universal approach, independent from the productive and technological development and from the public policy objectives of the regulating country.

Points 1 to 5 above, illustrate some of the ways in which the balance referred to may be struck. The clear limitation in the extent of protection (the expression and not the ideas or internal software structure), certain flexibility regarding the right to make copies and adaptations, a reasonable duration and the establishment of certain guarantees in favour of such users (such as the non-voluntary license provided for in the Republic of Korea), are among the elements that may contribute to attain such a balance.

As mentioned before, the number of developing countries that have already legislated on software protection is very limited. In many cases, the issue has not still emerged or gained public attention. In others, studies are only starting at the academic or governmental level. Finally, in a third group, pressures by the United States or by organised local associations (mainly those controlled by distributors of imported software) are pushing for the adoption — by legislation, administrative act or jurisprudence — of the copyright approach. In addition, the initiatives of the United States and other industrialised countries to define international "norms and standards" within the Uruguay Round include, among other matters, rules relating to computer program protection under copyright.

If the copyright scheme is imposed in GATT negotiations, the immediate consequences for most developing countries party to GATT would be the adoption of new laws and the amendment of existing ones in order to bring their intellectual property systems in consistency with the agreed norms. This would imply the loss of GATT concessions and advantages for countries unable or unwilling to adapt their legal regimes to the minimum standards and for those unable to enforce them.

In this context, most developing countries will be confronted, in a bilateral or a multilateral framework, with the need to decide on the software protection issue. Considerable room for co-operation among such countries seems to exist. That co-operation may take various forms and imply different degrees of commitment, ranging from co-ordinated action in bilateral and multilateral negotiations, to the definition of a more substantial common position[25]. Joint efforts to understand the implications of software protection and to devise the most appropriate legal models may therefore also be fruitfully envisaged[26].

In sum, the strategic options for NIEs and MIEs on software protection are limited by the newness of the issue and the ambiguities that still prevail on the form of regulation, as well as by the choice already made by the majority of industrialised countries. In view of the growing dissatisfaction with the parameters and results of protection through copyright, however, the best solution for many countries would be just to wait until a more precise picture is available. In fact, no real urgency to deal with the matter — at least from a legislative point of view — would exist in most developing countries, if the main concern is the protection/promotion of local software production. As said before, to the extent that custom software largely prevails, contract law may be a more effective mode of protection than a general regime.

In the event that bilateral or multilateral pressures make it necessary to produce a more immediate response, it should be clear that copyright is neither the best nor the "natural" solution, and that skilful drafting of pertinent rules may permit the right balance between the private and public interests involved.

VI. Main Conclusions

Any analysis on the software protection issue requires full understanding of the economic, institutional and technological context in which the debate takes place. Study of the world software market reveals a number of important facts in that respect: its dynamic growth; the overwhelming importance of developed countries as users and producers, the dominant position of United States firms; the high concentration of the supply, and the centralisation of R&D activities, among others. It also indicates that NIEs and MIEs have no comparative advantage for successful competition internationally, or even domestically with imported packaged software. If substantial efforts to improve production and marketing methodologies are not made, the participation of such countries in the software area may remain illusory.

The consolidation of copyright as the basic approach for software protection, cannot be attributed to its appropriateness for the subject matter. It rather shows the power of the country leading in software — the United States — to force the adoption of a legal system that basically reflects the interests of its industry — the most internationalised one among the OECD Member countries. The ambiguities and uncertainty that the application of copyright creates, has promoted the search for alternative forms of protection. Some of those initiatives were abandoned under United States' pressure. Others — like the application of patent law or the new approach suggested by OTA — indicate that even within that country the issue is still open to controversy.

Copyright protection of software is generally considered in developed countries as a means of promoting innovation and ensuring a reward for investments made in the development of new products. The attempt to transfer the legislative pattern adopted by such countries to the rest of the world, assumes that a similar legislation will have similar effects, independently

of the technological and economic context in which it will be applied. It seems clear, however, that the extension of copyright would mainly benefit software exporter firms that operate on a world scale. It is questionable that the protection would foster the diffusion and local software production in all countries, particularly in developing ones.

Diffusion may, in practice, be hampered by provisions such as those concerning adaptations and copying. However, since the total exclusion of protection does not seem potentially sustainable, the problem in that respect is how to strike a proper balance between producer, user and public interests.

From the point of view of production, local firms have not too much to benefit from protection if they are basically involved in the development of custom software, for which contractual law is the main means of protection. The production of packages may introduce a different picture, since it is not possible to compete with a low cost "pirate" industry. In any case, the impact of the legislation will depend on the segments in which local production will compete and on the terms under which the protection is granted.

It is clear, on the other hand, that mere protection is not sufficient to promote and give viability to a software industry. Other specific policies may be necessary in order to overcome the often serious obstacles that NIEs and MIEs face in this sector.

Finally, to the extent that the question is not whether to grant protection or not, but what type of regime is best and when it should be implemented, the regulation of different aspects (scope of rights, duration, etc.) is particularly relevant. From a technical point of view there is considerable room to frame a legal regime that takes into account specific diffusion or productive objectives, and which pursues a balance between public and private interests. The foreign policy implications of such a national decision on the subject are, of course, a different matter.

Notes and References

1. For western Europe, INPUT estimates that the market will grow between 1987 and 1992 at an average annual growth rate of 24 per cent (INPUT, p. 4, 1987).
2. The figures for the Brazilian software market contained in the OECD study, however, should be cautiously considered. Other sources estimate a considerably lower market size.
3. In Brazil, it has been noted, for instance, that though there are capabilities to develop an ADA-like compiler, skills for managing a project for the development of an environment in that language (which would involve a million and a half lines of code) do not exist (Pereira de Lucena, p. 17, 1988).
4. Software engineering tools are very rarely used in Argentina (SPCALAI, 1988). In the long term such tools may erode eventual competitive advantages based on the availability of low cost — qualified personnel.

5. Pressures have been exercised on several Asian and Latin American countries (particularly Brazil). Thailand is still in conflict with the United States on this matter (Krim, 1989).
6. After hesitation, countries of the former Soviet Union are also likely to join those countries who support the copyright approach.
7. Countries such as the Republic of Korea, which had not adhered to such conventions, have recently revised their position thereon, in part as a result of American direct pressures.
8. In France and the United States, on the contrary, a low originality requirement is applied.
9. See, however, the implications of the Whelan case below in this section (point c).
10. This fact explains that national producers concentrated on custom development, and did not discover the issue of software protection until pressures of package distributors emerged.
11. A similar decision was taken in the Gem Scan case in Canada.
12. This interpretation has also been embraced by the United States Copyright Office, though other decisions have ruled that a separate protection for such displays should be sought for (Russo and Hale, p. 9, 1988).
13. In accordance with one opinion, the protection of microcode by copyright could result in an extension of the monopoly of the copyright owner beyond the termination of any patents governing the computer systems. "The lengthy copyright monopoly with its presumption of validity would be a frightening weapon having significant 'in terrorem' effect against any competitor developing a computer with an instruction set compatible with a previously developed computer or microprocessor, whether copied or not" (MacPherson et al., p. 4, 1986).
14. Examples of patented software inventions include a process for a management control system for multiprogrammed data processing, a method of constructing a task program for operating a word processing system, a program that checks for spelling errors, and a program that converts one programming language into another (an RPG to COBOL compiler). Perhaps the best known software patent was issued to Merrill Lynch for a Securities Brokerage and Cash Management System. Protection is conferred by the PTO without requiring the submission of full scope-program, i.e., only partial disclosure is being accepted at the administrative level.
15. The paper was prepared by the Department of Communications and by the Department of Consumer and Corporate Affairs, Supply and Services.
16. In *IBM Corp. vs. Ordinateurs Spirales*, a Canadian court, however, accepted copyright for an object program. In 1988 the copyright law was amended in order to fully incorporate software as a copyrightable work.
17. Arguments for a new form of legal protection in the United Kingdom, with a shorter term and tailor-made rules are presented by Staines (1988).

18. In its proposal on intellectual property in GATT, the EEC has held, for instance, that software protection should take account "of legitimate interests of users, the promotion of international standardization, the development of compatible and inter-working systems and maintaining the conditions of competition" (July, 1988).
19. In Thailand, for instance, Lotus 1-2-3 could cost US$ 715, more than twice as much as many Thais earned in a month (Krim, 1989).
20. The Argentine survey was responded to by 156 firms producing, importing or distributing software (Subsecretaria de informatica y Desarrollo, p. 72, 1987); in the Republic of Korea, 384 replies were obtained on the basis of a questionnaire sent to 2,780 persons including businessmen, academics, researchers and public officials with ties to, or interest in, the computer software field (Song, 1987).
21. The weakening of the competition that may result from the application of a doctrine such as that held in Whelan is discussed in Bulkeley (1986).
22. In the United States, for example, though the CONTU report proposed to allow the right of copying to any "authorized possessor", the law restricted it to any "authorized owner" of a copy (Meisner, p. 394, 1987).
23. Among the comments and proposals made to the request of the Government of China (Taiwan Province) after the amendment of the copyright law in 1985, the establishment of a compulsory licensing system was suggested. "Under such a system, whoever needed a program could use it lawfully at a reasonable price. The software rightholder could avoid litigation expenses involved in pursuing pirates", (Chang, p. 464, 1987).
24. Limitations on the "moral rights" of a program title holder may also be found in the legislation of France (Correa et al. 1987).
25. The so-called "Group of Eight" Latin American countries, for instance, has agreed to co-ordinate their positions in GATT negotiations on new areas, including intellectual property ("Acapulco Declaration", 1987).
26. Representatives of Parliaments of twelve Latin American countries recommended, in 1987, the preparation of "a model of informatics legislation for the (Latin American) region". See Informatica e Integracion en America Latina y el Caribe, p. 19, 1987.

Bibliography

American Programmer, "Case Tools from Singapore", Vol. 1, No. 7, September 1988.

Associación Nacional de la Industria de Programas de Computadoras (ANIPCO), "Oportunidades para el desarrollo tecnológico y comercial de la industria del software en Mexico", Mexico, 1987.

Bertrand, A. and M. Couste, "Current Issues Concerning French Software Protection", *Law & Technology Press*, Vol. VI, No. 12, May 1988.

Borking, J., "Results of a Socio-Legal Survey Regarding the Legal Protection of Software", *Law & Technology Press*, Vol. VI, No. 6, November 1987.

Bulkeley, W., "Courts Expand the Copyright Protection of Software, but Many Questions Remain", *The Wall Street Journal*, 18 November 1986.

Burgess, J., "The Battle over Software Protection", *The Washington Post*, 2 February 1989.

Business Week, "Editorials: Don't Use Copyright to Shackle Software", 29 May 1989.

Chang, C.N., "Computer Software Protection in Republic of China (Taiwan)", *Computer Law Journal*, Vol. VII, No. 4, Fall 1987.

Computer System News, 1985.

Computers Today, New Delhi, 1988.

Correa, C.M., "Comercio Internacional de Software", Subsecretaria de Informatica y Desarrollo, Buenos Aires, 1987.

Correa, C.M., "Computer Software Protection in Developing Countries: a Normative Outlook", *Journal of World Trade*, Vol. 22, No. 1, 1988a.

Correa, C.M., *Propriedad intelectual, innovación tecnológica y comercio internacional*, Centro de Economia Internacional, Buenos Aires, 1988b.

Correa, C.M., et al., *Derecho Informatico*, Ed. Depalma, Buenos Aires, 1987.

Dataquest, "Birth of the Indian Software Industry", New Delhi, January 1987.

Desjeux, X., "Logiciel, originalité et activité créative dans la loi du 3 juillet 1985", in *La protection des logiciels sous la loi du 3 juillet 1985*, Ed. des Parques, Paris, 1986.

Flamee, M., "Aspects actuels de la protection juridique du logiciel au regard du droit belge", *Revue de Droit Intellectuel d'Ingenieur Conseil*, November 1985.

Forero Pineda, C., *Informática e integración económica*, Tercer Mundo Ed., Bogota, p. 49, 1989.

Fraser Mann, J., "Computer Programs and Copyright: Recent Canadian Developments", *International Computer Law Adviser*, January 1987.

Higashima, T., "A New Means of International Protection of Computer Programs through the Paris Convention. A New Concept of Utility Models", *Computer Law Journal*, Vol. VII, 1986.

Informática e Integración en América Latina y el Caribe, *Boletín de la Secretaría Permanente de la Conferencia de Autoridades Latinoamericanas de Informática (APCALAI)*, No. 23, Buenos Aires, June-July 1987.

Informática e Integración en América Latina y el Caribe, No. 27, Buenos Aires, 1988.

INPUT, "The Western European Market for Information Services. Analysis and Forecasts, 1987-1992. Executive Overview", London, 1987.

Jonqueres, J., "The Patentability of Software", IIC, No. 5, Munich, 1987.

Katz, R.L., La industria del software en los Estados Unidos; estructura y comercialización de producto, Conferencia de Autoridades Latinoamericanas de Informática (CALAI), Buenos Aires, 1987.

Krim, J., "Thailand's Refusal to Protect Copyrights Produces Cheap Goods, Disputes with United States", *The Washington Post*, 13 March 1989.

Levenfeld, B., "Israel Considers Comprehensive Computer Law", *International Computer Law Adviser*, March 1988.

Mac PhersoN, A. et al., "Microcode: Patentable or Copyrightable?", *European Intellectual Property Review*, p. 3, 1986.

Maier, G., "Software Protection — Integrating Patent, Copyright and Trade Secret Law", *Journal of the Patent and Trademark Office Society*, Vol. 69, No. 3, 1987.

Meisner, M., "Archival Back Up Copying of Software: How Broad a Right?", *Rutgers Computers & Technology Law Journal*, Vol. 14, 1988.

MITI, Information Industry Committee, "Aiming Towards Establishment of Legal Protection for Computer Software", Tokyo, 1983.

OECD, *Software: an Emerging Industry*, Paris, 1985.

OECD, *Internationalisation of Software and Computer Services*, Paris, 1988.

OTA, *Intellectual Property Rights in an Age of Electronics and Informatics*, Washington, 1986.

Pereira de Lucena, C.J., "A tecnologia de software no Brasil: a caminho de uma participação no mercado internacional". Paper prepared for the Centre de Estudos em Política Científica e Technológica do Ministério de Ciência e Tecnologia, 1988.

Russo, J. and T. Hale, "Developments in Copyright Protection of Computer Software", *International Computer Law Adviser*, January 1988.

Sandison, H., "NEC Corp. vs. Intel Corp.: United States Court Finds Intel's Microcode Copyrightable", *European Intellectual Property Review*, p. 25, 1987.

Schware, R., "Software Industry Development in the Third World: Policy Guidelines, Institutional Options, and Constraints", *World Development*, Vol. 15, No. 10/11, 1987.

Shipley, G., "Computer Software Copyright, the Same but Different?" *European Intellectual Property Review*, Vol. 7, No. 11, November 1985.

Siegel, D. and Laurie, R. "Beyond Microcode: Alloy vs. Ultratak — The First Attempt to Extend Copyright Protection to Computer Hardware", *The Computer Lawyer*, Vol. 6, No .4, April 1989.

Song, S.H., "Protection of Computer Software in the Republic of Korea", WIPO/IP/JK/87/6, January 1987.

SPCALAI, "Producción de software en la Argentina. Calidad, ventajas comparativas y exportación de productos" (preliminary version), Buenos Aires, 1988.

Staines, A., "Why Copyright is Wrong for Programs", *The Financial Times*, 21 July 1988.

Subsecretaria de Informatica y Desarrollo, "Produccion y comercio de software en la Argentina", doc. SID No. 35, Buenos Aires, 1987.

Takahashi, T., and C. J. Pereira de Lucena, "A tecnologia do software no Brasil, Problemas e perspectivas" (preliminary version), Paper presented to the Seminar on Production and Commercialisation of Software, CALAI, Buenos Aires, September 1988.

Ulmer and Koll, E., "Copyright Protection of Computer Software", IIC, Vol. 14, No. 2, 1983.

UNCTC, "Informe sobre las estrategias y politicas globales de las empresas transnacionales en la industria de computación: consecuencias para los países en desarrollo", New York, 1984.

UNIDO, "*Software Industry: Development Approach*'", *by S. Yu and Y. Kim, ID WG.478/1 (SPEC), 1988.*

United States Department of Commerce, *A Competitive Assessment of the United States Software Industry*, Washington, 1984.

Wells, L., "United States Pressures on Indonesian Intellectual Property, Investment, Trade and Immigration Policies", FR/117/87/13, 29 January, 1987.

V

Implications for Developing Countries

The Potential Role of Software in Enhancing the Competitiveness of Developing Country Firms

Atul Wad*

Introduction

There is a fairly wide and expanding literature on the potential role of developing countries in the global software industry and on the related problems of developing a software industry in these countries.[1] Most of this effort has focused on how developing countries, particularly the Newly Industrialized Economies (NIEs) might be able to participate in the global software industry, which niches they could penetrate, and how they could take advantage of their comparative advantages in this regard.

Also addressed are the practical problems involved in developing local software production capabilities and acquiring the requisite know-how, technology and market intelligence and access.

Strangely, what has received relatively little attention is the potential role of software applications, from whatever source, for improving the productive efficiency of enterprises in developing countries, thus contributing to economic development and to competitiveness in international markets. Even though there are many examples of software applications making significant contributions to productivity in both the public and private sectors in developing countries in various sectors, no systematic study and assessment of these experiences have been made.[2]

In the modern global economy, international competitiveness is a driving concern. The trend towards freeing up domestic economies, liberalizing markets and encouraging private sector development and entrepreneurship has made it vital for firms everywhere to become more competitive on a global basis. For the developing countries, this presents a special challenge. After years of following "import substitution" models of industrial development[3] and a relatively protected domestic market, firms in developing countries are having to face the realities of hard competition both foreign and domestic. A major hindrance in this regard is the low level of efficiency of industry in the Third World.

* Director, Technology Programs, International Business Development, Northwestern University, Evanston, Illinois, United States.

In a general sense, there is a vast gap in productivity levels between industrialized and developing countries as measured in terms of gross output per worker or value added per worker[4], partly as a result of technology gaps. However, given a certain level of technology, it is also true that productivity is lowered by the inefficient utilization of technology and by inefficiencies in the management and organization of production itself.

Attempts have been made to capture these aspects of inefficiency, at least at the macro-level, through such measures as domestic resource cost (DRC), effective rates of protection (ERP) and total factor productivity (TFP). These measures, however, do not offer any useful insights into the likely causes of these inefficiencies. Pack[5], for example, argues that these measures do not tell you whether any possibilities exist for improving productivity, through, for example, skill transfer. The sources of inefficiency are not clarified in any manner.

On the other hand, recent developments in the management literature have highlighted the important role of "softer" procedural and organizational improvements for enhancing overall productivity. Distortions and "disconnects" in the manufacturing process, communications, inter-departmental linkages and information flows are identified as some of the root causes of inefficiencies in the system as a whole. The "new" management techniques, such as Just in Time, Zero Defects and Total Quality Management stress this focus on the processes that are associated with manufacturing and its relevance to improving the productivity and competitiveness of the firm. Most of this literature has focused on the industrialized world, though some efforts have been made to examine the implications of the new management practices for developing countries[6]. In recent times, this approach has taken an explicit focus on the technological processes of the firm and how these could be improved upon.[7]

Thus, there have been recent developments that have begun to address the issue of productivity and competitiveness in developing countries. A wide range of firm level and policy measures need to be considered in order to effect productivity improvements. Software can play an important role in this regard.

The purpose of this paper is to examine the practical issues involved in developing and applying software for the improvement of the competitive position of firms from developing countries, with a specific focus on small- and mid-sized firms. Furthermore, the paper is primarily concerned with the applications of software to the industrial sector in developing countries.

The paper will first discuss the broad changes in the global economy and how they are changing the concept and determinants of competitiveness in the modern world. It will then examine the potential role of software applications in improving the efficiency of domestic enterprises, and the different areas where software can have an impact. Following this is a discussion of the issues involved in developing or acquiring the needed types of software and the issues involved in the delivery of software to the users. The paper concludes with an analysis of the policy implications for developing countries

The Changing Global Context

Several dramatic changes in the global context for development make it more important for developing countries to become active participants in the world economy. The new global context presents a different set of challenges for these countries:

- A heightened pace of global competition and new bases for competition.
- The rise of the NIEs and the gradual entrance of eastern Europe into the global economy.
- An accelerating rate of technical innovation and change, particularly in the advanced technologies (*e.g.* informatics, biotechnology).
- The emergence of science-based knowledge-intensive technology, and its spread throughout the industrial world.
- The internationalization of capital and production and the globalization of manufacturing. The development of niche markets and product and market differentiation on an unprecedented scale, at least in the industrialized nations.
- Changes in the concepts of efficiency and productivity (from economies of scale to economies of scope), which places greater emphasis on flexibility and innovativeness.

These changes have had important implications for what constitutes competitiveness in today's world. With the shift to a new techno-economic paradigm, the traditional bases of competitiveness are being eroded and replaced by a new set of concepts based on quality, responsiveness, speed to market, flexibility and efficiency in service. In this new playing field, the traditional strength of the third world, cheap labour, may not retain the value it once had. On the other hand, the changing situation could open up new areas of opportunity for the developing countries.

The Challenge for Developing Countries

For developing countries, the implications of these changes are complex and serious. To compete, they must undertake a series of steps. They will have to:

- Improve the process of technological development and capability building in their industries so as to make them more productive and competitive. This is particularly true for countries that have undertaken policies to liberalize their economies and privatize public sector corporations. In order to survive and succeed in the more open and, hence, more competitive environment being created, local firms need to upgrade their technological capabilities through internal technology development, sourcing of foreign technologies, adaptation, joint-ventures, and other means.

- Developing countries must develop their international business strategies and increase their participation in global markets on the basis of their competitive advantages. This requires more systematic and sophisticated marketing, market "intelligence" and market access. Their firms need to be able to define "niches" and windows of opportunity, enter into cooperative arrangements with firms in other countries, and deliver quality goods or services on time and at the right price.
- Firms in developing countries must improve the quality of their manufacturing processes and their products, to be competitive. Inefficient modes of operation that were affordable under more protective environments need to be rectified. New management techniques and other measures are needed to improve productivity.
- Developing country governments must adopt innovative policies and institutional mechanisms to create conditions which foster productivity and make firms more competitive.

Clearly, these are not easy tasks. Even though there is a proliferation of new product and process technologies, it is difficult to keep abreast of relevant developments and sources. As technology becomes more complex, it is difficult to assess the relative merits of each without some basic resident expertise. Gaining access to sources, and to the resources needed to adapt and modify these technologies to local conditions is also a problem. Furthermore, negotiating technology licenses is a complex process requiring legal, marketing and technical expertise often not easily available to firms in developing countries.

Similarly, developing a more sophisticated approach to marketing requires networks, contacts, various forms of expertise and resources that such firms typically do not have or cannot afford. It is costly to conduct market studies, to make frequent marketing trips overseas, and to find foreign distributors and collaborators.

Furthermore, local technological capabilities and resources are often inadequate to meet industry needs. Even where large local R&D systems have been established, these are typically not in tune with the needs of industry and lack a "demand driven" approach. The value of these R&D capabilities to local firms is questionable.

In other areas, such as marketing, quality control and management, local capabilities are generally weak, if not altogether absent. The culture and infrastructure of support service organizations and consulting firms in these "soft" areas, which are taken for granted in the United States, rarely exist in developing countries. Where they do exist, they are generally priced out of the range of most small and mid-sized firms. This situation is compounded by a general lack of experience with, and subsequent distrust of, consultants.

Clearly, there is a gap between the needs of firms in developing countries with respect to their technological capabilities and business development efforts, and the institutional and infrastructural context in which they exist. Their institutions often were set up to meet other priorities, or are based on a different view of the role of science and technology in economic development. They tend to be bureaucratic, non-market-driven and not responsive to industry because they are normally subsidized by the state. The protectionist policies that many countries pursued in the past have insulated local firms from the competitive pressures that would have generated technological innovation, increased productivity, more sophisticated marketing, and more aggressive movement into international business.

Carlota Perez[8] has summarized the main points of difference between the old and new paradigms and what they imply for the firm: (see Figure 1)

A new perspective on technology

At the core of the issue here is technology. For decades, the international community has focussed on how science and technology can contribute to the economic development of the third world. What has happened as a result of the change in the entire basis of manufacturing and the dynamics of the market, is a fundamental re-thinking of the concept of technology itself. While it is not the purpose of this paper to delve too deeply into this matter, it is useful to mention some of the main ways in which our notion of technology appears to be cl nging.

Technology is not just the hardware, but includes a soft dimension as well (i.e. technical know-how, skills and knowledge related to technical knowledge and the commercialization process, management, marketing, etc.). These "soft assets" can be crucial to the proper utilization of the hardware, and an entire specialty area is evolving in the field of management that deals with this aspect of technology.[9]

Technology should be viewed as a whole system, .ather than discrete elements, incorporating R&D, design, process and production engineering, maintenance, management and marketing. Thus technology is a complete package of processes and equipment that are all needed to ensure a technological capability.

- Technology is becoming increasingly knowledge-intensive, with a rapid increase in the knowledge intensity of production and a growing research intensity in the development of new technologies. In some cases, the initial investment costs simply to enter into a high-tech sector such as semiconductors, is prohibitive even by industrialized country standards. For developing countries, this suggests serious barriers to entry into certain areas of technology and the more relevant question is where they can find a place with respect to the industry based on this technology.

	CONVENTIONAL COMMON SENSE	NEW EFFICIENCY PRINCIPLES & PRACTICES
COMMAND AND CONTROL	Centralized command Vertical control Cascade of supervisory levels "Management Knows best"	Central goalsetting and coordination Local autonomy/Horizontal self-control Self-assessing/self-improving units Participatory decision-making
STRUCTURE AND GROWTH	Stable pyramid growing in height and complexity as it expands	Flat flexible network of very agile units/Remains flat as it expands
PARTS AND LINKS	Clear vertical links/Separate specialized functional departments	Interactive, cooperative links between functions along each product lin
STYLE OF OPERATION	Optimized smooth running organizations Standard routines and procedures "There is one best way" Definition of individual tasks Single function specialization Single top-down line of command Single bottom-up information flow	Continuous learning and improvement Flexible systems/Adaptable procedures "A better way can always be found" Definition of group tasks Multi-skilled personnel/Ad hoc teams Widespread delegation of decision making Multiple horizontal and vertical flows
MANNING AND TRAINING	Labour as variable cost Market provides most trained personnel People to fit the fixed posts Discipline as main quality	Labour as human capital Much in-house training and retraining Variable posts/Adaptable people Initiative/collaboration/motivation
EQUIPMENT AND INVESTMENT	Dedicated equipment One optimum plant size for each product Each plant anticipates demand growth Strive for economies of scale for mass production	Adaptable/programmable/flexible equip. Many efficient sizes/Optimum relative Organic growth closely following demand Choice/combination of econ. of scale, economies of scope or specialization
PRODUCTION PROGRAMMING	Keep production rhythm; use Inventory to accommodate variations in demand Produce for stock; shed labor in slack	Adapt rhythm to variation in demand Minimize response time (As in "just-in-time") Use slack for maintenance and training
PRODUCTIVITY MEASUREMENT	A specific measure for each department (purchasing, production, marketing, etc.) %Tolerance on quality and rejects	Total productivity measured along the chain for each product line Strive for zero defects and zero rejects
SUPPLIERS, CLIENTS, AND COMPETITORS	Separation from the outside world: Foster price competition among suppliers; make standard products for mass customers; arms length oligopoly with competitors The firm as a closed system	Strong interaction with outside world: Collaborative links with suppliers, with customers and, in some cases, with competitors (Basic R&D for instance) The firm as an open system

Figure 1: The New vs. the Traditional Paradigm:
Contrast Between two "Ideal Types" in Managerial Common Sense

Source: Perez, Op. Cit.

- Technology can no longer be regarded as a cost of modernization but as an investment towards long-term growth and competitiveness. The "supply" push approach to technology with its assumption that the key to growth is the injection of more technology and more science needs to be revised. Innovation is the key concept today, and is much more a demand driven notion. Technology is still an essential input, but the type of technology, how it is used, and how it is developed need to be addressed on the basis of what is the demand for the final results of that technology, demands set in the marketplace and derived from the needs of society.

- Technology is not simply a static input to a productive system, but rather a **variable** that can be manipulated. Thus, the same piece of equipment can be used badly or well — as a result of non-technical factors. Simply having the technology is not enough, it must be used well, and this requires an understanding of and capability in the management of technology at the enterprise level, and the development of sensible and realistic technology policies and plans at the national level.

- Technology encompasses not only what is commonly included under the term — i.e. developed country technology — but traditional bodies of knowledge as well. Increasingly in use as an alternative to "technology" is the concept of "knowledge systems", which may range from complex computer systems to techniques based on accumulated knowledge in traditional societies. For example, traditional medicinal remedies based on local herbs and roots, food preservation techniques, etc. The scope of what constitutes knowledge has thus expanded, and thereby also become more complex.

In short, the entire notion of technology needs to be revised and reconceptualized in order to take into account the changing realities of the present time. The improvement of productivity of manufacturing therefore needs to be centrally based on an appreciation of the benefits that can be gained by using technology well.

The Role of Software

The current situation for developing countries

While there has been a great deal of concern over the implications of the computer revolution for developing countries and the debate continues, much of this debate has centered on the hardware aspects of the technology. In the past, there was concern over the options available to developing countries to build local capabilities in this technology, an option that is fast dwindling as the costs of initial investment continues to rise and the barriers to entry, at least in the core areas become even tighter. (On the other hand, the costs of computer systems continue to drop, making the acquisition of these systems more feasible in developing countries.)

Several countries have, however, made in-roads in the ancillary industries, such as printers, disk drives, monitors, etc. For example, the total production of the largest hardware manufacturers in the third world, mainly the NIEs, exceeded $12 billion in 1988. For the manufacture of peripherals, the comparative advantage of cheap labour will probably continue to shape the pattern of movement of the industry through the third world.

On the other hand, in the software industry, the participation of the third world has been much smaller. The United States continues to dominate the global industry, which is estimated to expand to $340 billion by 1996, by growing at an annual rate of 20-30 per cent. But in this business, the third world barely accounts for a few percentage points, although there is evidence that the size of the software markets in some countries is increasing rapidly (Figure 2) and governments in some countries, such as India, are taking proactive supportive measures for the development of this industry.

At the same time, the industry itself is going through major changes:[10]

- a shortage of trained personnel, resulting in a situation where software costs now account for the major share of total system costs
- a global struggle over operating system standards
- a trend towards customized, integrated multi-vendor hardware and software solutions

COUNTRY OR AREA	1984 SOFTWARE	1984 SERVICES	1984 TOTAL	1987 SOFTWARE	1987 SERVICES	1987 TOTAL
Brazil	363.5	337.7	701.2	2,,186.2	2,,031.4	4,,217.6
Hong Kong	25	n.a.	n.a.	61	n.a.	n.a.
India	18.3	92.8	111.1	37.7	298.9	336.6
Korea, Rep of	40	20	50	107	40	147
Malaysia	20	n.a.	n.a.	67	n.a.	n.a.
Mexico	59	6	65	117	13	130
China	175	n.a.	n.a.	968	n.a.	n.a
Saudi Arabia	25	n.a.	n.a.	49	n.a.	n.a.
Singapore	27	21	48	71	59	129
China (Taiwan Prov.)	26	29	55	57	51	108

Figure 2: Software and Computer Services Market in Selected Economies
(Current 1984 $ Million)
Source: Schware, 1989, p.32

- a growing emphasis by hardware vendors on software production and sales, leading to a concentration of firms within the industry
- a simultaneous expansion and fragmentation of the industry resulting in a growing number of software vendors.

These changes could have serious implications for developing countries in terms of their role in the global software industry. It may open up new opportunities in niche areas, but it may also close existing options. In the case of "body-shopping", for example, which is practiced by companies from India, who sell cheap labour for software development through marketing arms in the industrialized countries, a common problem faced is that potential clients are uncomfortable with the arms-length transaction involved and prefer to deal with software developers who have their staff and production close at hand. This despite all the arguments of cheaper labour, ease of transmission of data, etc. There is concern over quality control, after sales service, protection of proprietary knowledge and delivery schedules. Nevertheless, several firms from developing countries have been successful in selling software services to clients in the North.

Software for domestic applications

The focus of this paper is on how software can contribute to the improvement of productivity of industry in developing countries. Issues related to the participation of developing countries in global software markets are closely related but distinct in this regard.

At a first glance, there seems to be a strong argument in favour of the potential for software for solving problems in the third world. Most developing nations are characterized by:[11]

- Low productivity
- Labour intensive manufacturing operations
- Underemployment
- Poor work habits
- Inadequate maintenance
- Low skill and education levels
- Material shortages
- Improper supervision
- Resistance to change

Damachi and Souder argue that it is precisely these types of problems that lend themselves to solution through the applications of computers, which are characterized by:

- Low equipment costs and declining trends in costs
- Cost effective operations

- Remove or reduce repetitive tasks
- Enable productivity control
- Assist in quality control
- Can be made user friendly
- Improve the quality of data storage, retrieval and analysis
- Enable decentralized decision making
- Most systems are standardized

Thus, there is a strong and logical set of arguments as to why software can be used to enhance productivity in developing countries. On the other hand, the diffusion of software applications has generally been slow. Partly, this is a function of the slow adoption rate of computer technology in general. In most developing countries, the large corporate sector and public sector undertakings have been the driving force in generating a demand for computer technology, but small and medium sized enterprises have not been as active. Yet, it is in the small and medium sized sector that some of the most valuable benefits could be reaped.

Barriers to Software Development

There are several reasons for this situation, some related to the characteristics of developing countries themselves, and some to the nature of software applications.

The barriers to the adoption of computer technology in developing countries include:

- Poor infrastructure and communications systems, which impedes the proper utilization of the technology
- Lack of appreciation by end users of the benefits to be gained.
- Perceived high costs associated with computers, even though this is increasingly not the case.
- Lack of financial resources, and foreign exchange to purchase the required technology
- Concerns over the security of computers, especially for small businesses which often use innovative accounting techniques.
- Shortage of trained personnel to operate the equipment.
- Antiquated management philosophies and systems which do not lend themselves to the adoption of computers. Many managers in developing countries feel threatened by computers, because they fear loss of control and a reduction in their own status in the organization.

- Lack of information about the types of applications possible with today's computer technology.
- Only a few local companies with the resources and capabilities to deliver software applications effectively. Many are simply vendors of imported software and do not have the capacity to provide on-site assistance in installation, trouble-shooting and problem-solving. There is no "hot-line" for software users.
- High costs of the software itself, because of licensing fees, royalties, duties and high profit margins.
- In some countries, a situation where hardware has been oversold, and has been under-utilized, leading to a general mistrust of the technology, and a hesitance to invest further in software that would in fact make the hardware more productive. There is little appreciation of what Schneider[12] refers to as "hardware+software" thinking
- Standard software packages do not fulfill user requirements totally.
- Inflexibility on the part of the vendors, who expect the organization to adapt to the software rather than vice-versa, which relates to the lack of local capability to adapt and modify software systems for local uses.
- The "body-shopping" syndrome, which results in the best talent being drawn to work on projects for foreign clients. The salaries tend to be higher, and there are other perks such as foreign travel.
- Improper government policies, which tend to favour exports of software services rather than provide incentives to local companies to develop a domestic client base. Also, in many countries, the policies tend to emphasize the "supply" side rather than attempt to foster a real local demand. Thus, in India, the government encourages the training of software engineers and rewards companies that are able to sell services overseas.
- Lack of experience in the marketing of software in developing countries. Firms tend to under-invest in the marketing effort, even though even a cursory analysis of the industry indicates how important it is to success.

Furthermore, there is the larger problem associated with what is referred to by Schneider as "polluted software", the fact that nearly all software systems in developed countries are badly structured and difficult to maintain. He even goes so far as to state:

"The state of the art in data processing in industrialized countries is not worth imitating. Through rationalization of hardware production, the hardware/software cost ratio is now 20:80 and is predicted to be 10:90 in 1988. Programming staff in industry and in public administration devote 80-90 per cent of their total manpower to software maintenance. The

value of installed software worldwide is about $150 billion to $200 billion. Nearly all these software systems are badly structured and difficult to maintain. This is called "polluted software".[13]

Thus we have a range of problems associated with the diffusion and adoption of software applications in the third world, both as a result of constraints specific to the developing countries, and as a result of the nature of the software industry itself. Yet, in a fundamental sense, software can make enormous contributions to productivity improvements in industry, banking, transportation, services, public administration and communications.

Schware, for example, cites several examples of successful software applications in developing countries :

- A hospital information system in Thailand
- A municipal management model that projects cash flow requirements in Brazil
- An accounting system for businesses operating in hyper-inflationary environments in Argentina
- A system for monitoring foreign exchange transactions jointly developed by Indian and British collaboration.
- a design and drafting package that incorporates CAD and CAM in Singapore
- Adaptation of standard software packages for Arab countries developed in Tunisia
- Several examples of text processing software in different languages — Korean, Thai, Chinese, etc.

The possibilities do exist. The question is how to overcome the barriers in order to speed the proper development and utilization of software in developing countries.

A balanced path for software development

The problem in the past with the manner in which the development of a software industry has been approached in developing countries is that is has been skewed towards:

- an emphasis on exports of software services
- a bias towards the newly industrialized countries[14]
- a lack of a balanced approach

The four broad categories of application of software in developing countries are:[15]

- Applications for basic needs and the agricultural sector
- Applications in the government sector

- Applications for productivity improvement in industry and service sectors
- Development of trade and export of software packages and services

In order to develop a suitable software industry, a balanced approach that takes into account all four of the above areas of application and their inter-relationships is required. This entails understanding how export trade is linked with domestic demand and how the different elements of domestic demand relate to each other.

In addition, a balanced approach must take into account the types of priorities and constraints relevant to the development of these countries. While there are wide variations between developing countries, some generalizations can be made:

- A need to expand exports in order to earn foreign exchange for the acquisition of technology and other inputs to the economy. This is a pressing need for most developing countries, but with the exception of the NIEs, the export performance of most of the South has not been very encouraging in recent years, partly due to declines in commodity prices and partly due to increased competitive pressures. For some countries, there is the added burden of heavy external debt that further encourages a strong need to export. As far as exports of software services are concerned, this possibility only exists for a handful of developing countries, those with large pools of trained human resources. For the most part, developing countries face shortages of trained personnel in the software sectors. **However, they may have export opportunities in other sectors which could be expanded by improving the competitiveness of these sectors or sub-sectors through the proper application of productivity improving techniques, including software.**
- A need to address pressing social and economic problems — health, employment generation, income generation, nutrition, education and the over-arching problem of environmental protection. Again, software can play a productive part in this context — by enabling the improvement of the quality of social services, administration of health care, education and skill upgrading, etc. For the group of countries categorized as "Least Developed" by the United Nations, these problems are perhaps the most pressing.
- A very large role played by the state machinery in all sectors of the economy. This has two implications: first, the government is often the largest market for goods and services in the country and as such cause its own huge purchasing power to influence industrial development in desired directions. In many countries, especially India, China and Brazil, the computer industry has received a strong impetus from the public sector, accounting for over 50 per cent of total output of the industry in India for example. Secondly, the government bureaucracy is itself a source of administrative inefficiency that has an adverse impact on the rest of the economy. Proper applications of software in large government bureaucracies could improve their own efficiency and thereby improve the

environment within which industry functions. In many developing countries, it is the government bureaucracy that is often cited as the main obstacle to industrial growth — the minor problems and delays associated with getting permits and approvals, the enormous paperwork involved in importing equipment or obtaining foreign exchange, the complexities of the regulatory system etc. all militate against the efficient working of industry in these countries. While software will not be able to solve all of the inherent problems in third world bureaucracies, it would certainly help.

- A trend towards liberalization of the economy. Spurred by the initiatives of the World Bank and the International Monetary Fund in the 1980s, many countries have enacted Structural Adjustment programs aimed at developing market based economic systems and opening up their domestic sectors to international competition. The results of these efforts have been mixed, with the Least Developed Countries perhaps faring the worst as a result of poorly designed and implemented programs. Inertial tendencies from the times of protected domestic markets, along with a general lack of capability to compete effectively in international markets have had adverse impacts on some countries. Others, however, have continued to aggressively privatize their public sector corporations with some success, for example, Mexico. In all cases, however, there is now increased pressure to become more competitive and efficient simply in order to survive the new economic climate. There have been positive developments on the "input" side as a result of liberalization, as countries have gradually relaxed barriers to imports of technology and products in a variety of industrial sectors, including the computer industry. On the other hand, the private sector continues to be relatively weak in most countries, whereas it could be the most dynamic and innovative.

- Finally, there is the common concern in all developing countries with the development of endogenous capabilities, however defined, so as to be able to pursue self-reliant and sustainable paths of development. More specifically, the concern is focussed on developing technological capabilities so as to be better able to develop, acquire, adapt and use technology for development. This need applies as much to software technology as it does to other areas, and the development of long term capabilities in software technology and know-how needs to be a central consideration in any policy making exercise. We shall return to this issue in the next section.

Developing a balanced approach to software in developing countries therefore needs to be based on a number of considerations:

- The generation of both a demand for software services, and the enhancement of the supply of inputs, such as trained personnel, for the growth of the industry. Demand generation is a role that the state can and should play through its purchasing policies and various incentive schemes. Training programmes and relaxed import regulations can support the supply side.

- A balance must be achieved between the concern over exports and the generation of a domestic demand and capability to meet this demand. In the long run, an export capability cannot sustain itself without a strong domestic base — this is a lesson learned in many industrial sectors and is indeed a key element in determining the competitive advantage of a country in a particular industry. An overemphasis on exports can in fact lead to an "enclave" situation, where the export oriented industry has very few linkages within the domestic economy. In the extreme, this industry could price itself out of range of domestic firms and this may already be happening. For example, even though Indian software engineers are inexpensive by world standards, they are very highly paid by Indian standards. A software engineer in India with the two to four years experience earns as much as someone with 15 years of experience in other fields. This reflects itself in the cost of software services to Indian industry, and makes software affordable mainly to the large corporate sector or the government. Small businesses are unable to afford these services.

- The development of local capabilities for the long term in the appropriate areas of software engineering. Since the global industry is itself undergoing a major change, it is not clear exactly where efforts should be focussed. Imitative strategies will not work for a number of reasons, not least of which is that the path that the software industry in the North has followed is itself being questioned. Some of the problems that software will be called upon to resolve include:[16]

  - problems that are not amenable to algorithmic solutions
  - problems that involve judgmental decisions
  - problems that require very context specific knowledge that must be consulted dynamically, for example, medical diagnostics
  - problems where the solutions cannot be specified beforehand, but must evolve in an open-ended fashion.
  - problems that involve the integration of a mixture of system components and are based on inadequate or poor quality data.

Furthermore, the economics of software production are changing, with an increasing emphasis on software quality, productive efficiency, economies of scale and market intensity, increasing intensity of R&D (which is itself a barrier to entry), and the lack of availability of financing and venture capital for new start-up firms.

As the technology of software develops, new possibilities are opening up, and new demands are being made upon the technology. For example, there is increasing interest in the potential of process control and monitoring software, based on expert systems, to address environmental pollution concerns in small firms in the United States. In fact, the increasing concern over the environment is raising a new set of challenges for the industry world-wide. On the hardware-side, many of the manufacturing processes associated with computers are now being questioned for their environmental soundness. On the software-side, the poten-

tial for solving pollution problems at all levels, in industrialized and developing countries, is being explored. Software packages have already been developed to anticipate toxic spills from factories and avoid the chances of reoccurrence of a Bhopal-type incident.[17]

An important recent development is "knowledge-based engineering", which is a soft are technology that, "provides a means of storing a product or process attributes, rules and requirements. The rules and requirements can generate designs, tooling or process plans automatically".[18]

Unlike traditional CAD software, knowledge-based engineering systems capture the intention behind product design and provide a richer and more flexible design tool. Companies such as Eastman Kodak and General Electric have been able to cut down design times by orders of magnitude with this software. In an age when "time to market" and "design lead times" are crucial to achieving a competitive edge in the marketplace, the value of knowledge-based engineering cannot be underestimated. For the third world, where design times are typically much longer, the application of an appropriate form of this software could yield significant competitive benefits in three respects:

- reduction in time to market
- leveraging of existing engineering knowledge
- improved capacity for concurrent engineering

At present, knowledge-based engineering systems require fairly large and powerful systems, but with advances in hardware technology it is not unlikely that they will be able to reside on smaller work-stations or even PCs in the near future.

All of this needs to be seen within the context of the emerging new paradigm of manufacturing itself, discussed earlier. In fact, the issues related to the potential and role of software in developing countries reflects the problems and issues facing the developing world in general as it addresses the challenge of how to develop in a new global context where competition and productivity have become critical requirements. Software technology can make a significant contribution to manufacturing efficiency in developing countries.

Technical change and technological capabilities: the emerging perspective

The question of developing domestic technological capabilities in software has been discussed earlier. Much has been written about the general problems of technological capability development in the third world Recent developments in research on technical change have however shed a different light on what constitutes technological capability Before turning to the specific question of what technological capability in software involves, therefore, it is useful to briefly review the changing concepts of technical change and in particular technology choice

Technology choice is no longer as simple as the neo-classical view would suggest. Other factors must be considered in the process, including the current economic conditions in the South, the structure and efficiency of the market, issues of uncertainty and incompleteness of information, the role of management and other "soft" assets, and the evolutionary nature of the technical change process itself. Research by a number of economists, including Richard Nelson and Sydney Winter, Giovanni Dosi, and, for the developing countries, Jorge Katz and Sanjaya Lall, have developed this perspective and provided a new conceptual tool kit for the analysis of technical change.[19] This growing body of literature suggests that technological change does not occur in the steady, linear and incremental way presented in traditional economic models. These models assume that technological change occurs as a spontaneous and linear response to signals from the marketplace; that the market functions perfectly and spontaneously, such that these signals precisely reflect a demand for technological innovation that will always provide a competitive edge to any company that can meet the demand; and that all market actors will read the commercial environment, and react to it, in the same way.

These assumptions of perfect response to market signals, perfect market functioning, and homogeneity of all market actors are gradually being replaced by a new model of technological change, which is less linear and leaves more room for interpretation, but which fits better with the process of technological change as it actually happens. This model stresses factors that control the diffusion of technological innovation, such as the tacitness of technical know-how (i.e. the difficulty in codifying such knowledge completely), the appropriability of the returns from innovation and the important role of learning. It more accurately accounts for the complex interaction between all stages of the innovation process, from basic research to commercialization, and better explains the wide variations in patterns of technological change in different industry sectors. Given these differences to traditional economic models, this new model is more demand-driven and stochastic. With reference to the search process, it emphasizes how firms are likely to search first at the boundaries of their own knowledge before moving too much further, how the uncertainty and lack of information associated with new technology may steer them in directions that are "safer" rather than more innovative, and how over time, firms move in directions that allow them to accumulate learning and know-how in an evolutionary fashion.

This perspective also puts a needed stress on the "soft" side of technology, pointing up the fact that technological change is not simply a matter of installing new hardware. New hardware brings with it a host of demands for new learning and ancillary technologies needed to maximize the value of the new equipment. This perspective is especially important in the area of computer technologies, which tend to be whole systems involving software, hardware and service.[20] It is also important in the sense that it is precisely in optimizing this "soft" side of the manufacturing process that software can play a vital role.

The "soft" side of technology is receiving increasing attention in the literature. In a recent review of the literature on flexible specialization and third world industrialization, James and Bhalla note the increasing emphasis being given to organizational innovation. The empirical base for this trend in the developing countries, however, remains small. However, they cite a recent study of the garment industry in Cyprus, which showed that three changes — reorganization of the production line, introduction of a computerized information processing system and the closer integration of marketing and production — contributed to a substantial improvement in the competitive position of the firm.[21]

There are other reasons why the "soft" side of technology may be particularly suitable for developing countries to try to improve. Pressures to lower cost and reduce waste, to manage inventories, reduce design lead times, and satisfy a more differentiated market can encourage the adoption of organizational innovations. The relatively lower costs of introducing such innovations, over hardware, is also to be considered.

This admittedly more complex model is nonetheless more accurate for understanding technology choice and innovation in the context of the weak and distorted markets, institutional weaknesses, and the lack of access to information that characterize many developing countries.

Developing country firms generally have weak domestic technology generation capabilities. This is particularly true in the software business. Therefore, the search for technology is likely to be primarily an external one that is limited and conditioned by what they know to be available. Hence the quality of the search process becomes an important determinant of the quality of technology choice. This capability is often referred to as "technology intelligence", or as "technology sourcing and intelligence"[22] and is a crucial component of overall technological capability in software.

The concept of "technological intelligence" is important for our purposes because it encompasses a wide range of capabilities with respect to the ability to effectively identify, acquire and use technology. It includes capabilities with respect to:

- the identification of technological needs at the firm, sectoral and national levels
- the assessment of available technology
- technology sourcing capabilities, along with the ability to monitor technological developments on a global basis
- the management of technology
- technology forecasting and impact evaluation

Developing such capabilities requires a detailed understanding of the entire technical change process as well as a range of practical initiatives directed at institutional development, skills, and policies. Furthermore, it must be based on a deep understanding of the constraints under which firms in developing countries operate, for example:

- poor information about technological alternatives
- shortage of skills in technology acquisition and adaptation
- financial constraints
- small domestic markets
- poor quality control
- bureaucratic constraints imposed by governmental policies and structures.

Technological capabilities for software

Drawing from this broad overview of the dimensions of technological capability in developing countries, it is possible to narrow down to the specific elements that appear to be most crucial to the development of such capabilities in the software industry. Three broad categories of capability can be identified as important for the software industry:

- Capabilities in monitoring and sourcing of software techniques and know-how on a global basis — *technology sourcing*.
- The ability to adapt and digest technology obtained from external sources for local purposes and to develop technologies locally — *technology adaptation*.
- The capacity to deliver the technologies to the end users effectively — *technology delivery*.

Of these, the most important with respect to software is the capability to source effectively. In the case of adaptation and delivery, the issues involved are more or less the same as with other areas of technology.

Technology sourcing

The effective sourcing of technologies by firms in developing countries is increasingly being recognized as vital to the long term development of technological capabilities. In the industrialized countries, corporations both large and small are appreciating the value of external sourcing of technology, and its importance to their competitive position. For developing countries, this is particularly important in fast moving areas of technology, such as software, which are also typically the areas where they have the weakest endogenous technology generation capabilities.

Technology sourcing is more of an art than a science. Practitioners in the United States each have their own particular approach and style. However, certain general principles can be identified that have proved useful in the sourcing process:

- A horizontal perspective on technology; applications in one area may have value in another.

- A global perspective; even though the United States dominates the software business, important applications may be available elsewhere.
- Sourcing is a cumulative process, not a one shot event; it must be an on-going effort on the part of the firm, even to the point of designating a person or persons with the sole responsibility for this activity.
- Personal visits and contacts are always more effective than arms length transactions. This is repeated by almost every professional involved in this business. The quality of information and the interest that is stimulated is much greater when a personal contact has been made.
- The search must be based on a careful assessment of software needs of the domestic industry; it must be based on market "intelligence" and technology needs assessment. Prior to starting a search for technology, a demand survey is desirable. The types of questions that need to be answered are illustrated in Figure 3.

In addition, software firms may find it useful to collect background data on the firms they see as potential clients and use this to do their own analysis of the types of needs of these firms that could be satisfied by appropriate software. A typical listing of the type of information that would be needed to identify the specific software needs of a firm is shown in Figure 4.

The purpose of this information is primarily to be able to assess where there may be room for improvement in the performance of the firm and how this could be accomplished through software introduction. For example, the quality related questions may reveal that the company has a poor image because of a high level of returns, which is in turn a function of poor internal quality control, which could be improved by introducing a total quality management package to the company.

The above discussion reflects to some extent the real experiences of some software companies in developing countries.

Lessons from an Indian Software Company

One of the largest Indian software companies has stressed the sourcing of new products and technologies for years, with considerable success. Based on interviews with a senior executive from this company, the following general lessons about sourcing were drawn:[23]

- Sourcing as such is not a difficult task, but is labour and time intensive
- Fairs and expos around the world tend to be an important source of information, and travel to these shows on a regular basis is essential.

- Of the three main regions for sourcing, South-East Asia, Europe and the United States, South-East Asia is good for low end hardware needs, Europe for medium-level hardware needs, and the United States for software. Europe is a good source for customized software and South-East Asia has little to offer in software.
- The demand for software exists in the domestic market. The needs have to be understood, though. This can be assessed through surveys or other channels — trade journals, manufacturers associations, feedback from the sales force, and

1.What are your major software needs at present?
- — management applications (accounting, finance etc.)
- — spare parts and inventory control
- — production (e.g. MRP)
- — quality control
- — service and maintenance
- — other

2. What business goals will this technology satisfy?
- — new products
- — improved quality
- — exports
- — reduced costs
- — larger volumes
- — customer relations
- — new markets

3. How do you normally identify the sources for software?
- — personal contacts
- — literature
- — trade shows
- — research institutes
- — universities
- — consultants
- — sales calls by equipment manufacturer

4. What factors dc you consider in selecting a specific software package?
- — cost
- — after sales service
- — reputation of supplier
- — equality
- — compatibility with existing software
- — ease of use
- — level of training needed
- — cost of maintenance and trouble-shooting

Figure 3. Questionnaire for Technology Needs Assessment

1. **Corporate Profile**

   Ownership
   Employees
   Sales (Annual)
   Products (Range of Products)

2. **Market Position**

   Market share
   Major competitors
   Distribution networks/channels
   % of sales for exports

3. **Technology**

   Type of equipment
   Age of equipment
   Source of equipment

4. **Production**

   Throughput (volume per hour or day of product)
   Number of shifts
   Raw materials
   Percentage waste/scrap
   Down time of equipment

5. **Quality**

   Product image among consumers
   Number of defective parts produced
   Average inventory
   Equipment failure frequency
   Quality of raw materials and other inputs
   Number of customer returns

6. **Human Resources**

   Management/employee ratio
   Skills profile and areas of weakness.

Figure 4. Diagnostic Survey for Assessing Software Requirements

from government sources.

- Generally, large customers know what they need, whereas the smaller companies are less clear and have to be persuaded.
- It is good to have a long shopping list when starting the search and sourcing effort since not all needs will be met.
- Keeping abreast of the technical literature is extremely important, especially when entering the final stages of negotiations for technology purchase.
- Part of the sourcing effort is to determine what products and technologies are working well in the North, since it enhances your confidence level.
- Having a clear-cut procurement policy and machinery is important to expedite the acquisition of the technology once it has been selected.
- The sourcing effort, as well as the adaptation process should be properly costed out and this cost should be reflected in the price charged to the customer. Firms tend to ignore these costs and often underprice a new product.

Finally, in the acquisition of technology, there is the entire issue related to negotiating. In the case of software, negotiating becomes particularly difficult because of intellectual property rights issues and the problems of evaluating the worth of what is essentially a product of "brain-power". The acquirer of technology needs to be able to negotiate with a full knowledge of the legalities, accounting issues and proprietary issues involved in order to get the best deal. Strengthening negotiation capabilities for software acquisition is an important component of the technology sourcing capability of a firm.[24]

Technology adaptation and technology delivery

Acquiring the right type of technology is the first step, and must be followed by an efficient process of "digestion" and delivery of the technology. Firms in developing countries often spend long periods adapting and absorbing new software technology. Yet, this is a real cost to the firm and needs to be minimized. There are few studies that examine the processes that follow the acquisition of technology by a developing country firm, and as such our understanding of how this could be improved is limited. On the delivery side, the same issue arises. Adapting the software to the specific needs of the customer, training the user and providing efficient follow-up assistance and trouble shooting is essential to the successful utilization of the technology. Few firms in developing countries pay attention to this aspect, and often leave the customer stranded with a software package that he cannot use.

The software industry in Jamaica: an illlustrative analysis

It is useful to examine the specific situation of the software industry in a particular country and a recent study by Gillian Marcelle of the Jamaican computer services industry is useful in this regard.[25] The study is one of the few thorough analyses of a domestic computer industry in a developing country and provides an excellent micro-level assessment of the problems and issues faced by the industry. This section draws heavily on the Marcelle report.

Sourcing of technology by Jamaican firms tends to be heavily oriented towards the United States for both hardware and software. In the case of hardware, the larger corporations, such as IBM, ICE and Apple tend to dominate. In the software area, the sources tend to be: hardware manufacturers of operating systems, software specialist houses, and specialized applications suppliers. Jamaican firms also source software through large distribution houses in the United States. Throughout, the United States dominates, though some sourcing does take place from other countries, such as Belgium, Canada, China (Taiwan Province), Sweden and the United Kingdom. About 20 per cent of Jamaican firms source locally as well.

On average, the costs of sourcing were estimated as between 0.3 and 24 per cent of revenues for the firms studied. These included expenditures on:

- Technical support fees, training and trouble shooting
- Purchases of related information, manuals, training course, etc.
- Acquiring general technical knowledge in the information services field and keeping abreast of technological trends. Expenditures on conferences, trade shows, industry reports etc. would be included here.
- Subscriptions and fees for access to electronic databases
- Fees for training courses and seminars.

The average level of expenditure on technology acquisition is high compared to other sectors in Jamaica, indicating the relative importance of this function to the computer industry.

The report also examined the sources of technological know-how used by Jamaican firms to augment what they obtained from the suppliers of technology. Trade shows, conferences, technical journals and training programmes were important for this "general capability development" purpose.

In assessing the overall industry, Marcelle concludes that in Jamaica, "All of the technology acquisition activity and in-house technology development is aimed at the product development stage. Training courses, purchase of journals, trade literature and newsletters as well as attending conferences and trade shows all provide detailed product knowledge and equip the

Jamaican technologists with a working knowledge of information technologies. Armed with these two inputs, they adapt and modify received information and products to create versions of existing applications or to enhance products but not to **fundamentally alter their characteristics of capabilities (emphasis added)**."[26]

As such, even though the Jamaican industry has become very adept at identifying and adapting technology from overseas and staying current with the state-of-the-art, domestic technology generation capabilities remain weak. To be able to contribute substantially to the competitiveness of Jamaican manufacturing firms and the efficiency of the services sector, this capability is essential.

Conclusion: The Importance of the "Soft" Dimension of Technology

This paper has attempted to provide some insights into the importance of software for upgrading productivity in developing countries. It is at best a cursory analysis, since the empirical base for the application of software to domestic industry in developing countries is weak, and is very scant when this question is addressed within the context of the new techno-economic paradigm.

Nevertheless, the writing is on the wall — to improve productivity, firms all over the world are having to look at the "soft" dimension of technology — the intangible processes and linkages that are as important to efficiency as the equipment itself. In Japan, the appreciation of these "soft" assets has reached a very high level. In the United States, the larger firms are beginning to appreciate it. But in the third world, there is still a long road to travel.

For the software industry the challenge is simple to describe and difficult to accomplish: identify where domestic firms can gain the most from software applications; find the relevant software technology wherever it is, adapt it to the local needs and deliver it speedily and effectively to the users.

For policy makers, the need is to develop a more balanced long term approach to the promotion of the software industry, taking into account the complexities of the modern global economy. Most importantly, there must be a balance between export emphasis and the generation of domestic demand. Making it easier for local firms to access and acquire software technology and develop domestic capabilities are two areas where the State can play a very positive role, by relaxing restrictions on imports of technology and using its own sizeable purchasing power to shape demand. Policy makers also need to look at innovative organizational mechanisms that could be established to improve technology acquisition and commercialization — technology incubators, technology sourcing mechanisms, technology banks and clearinghouses, etc.

On the policy research side, much more empirical work is needed on the micro-level. How firms acquire technology, what determines their competitiveness, and how technology is digested and adapted are all processes that need to be better understood in order to more clearly articulate the proper role of software.

Notes and References

1. Robert Schware, "Software Industry Development in the Third World: Policy Guidelines, Institutional Options and Constraints", World Development, Vol. 15, No. 10/11, pp. 1249-1267, 1987; Carlos Maria Correa, "Software Industry: An Opportunity for Latin America?", World Development, Vol. 18, No. 11, pp. 1587-1598, 1990; Ukandi G. Damachi, H. Ray Souder and Nicholas A. Damachi, (eds.) Computers and Computer Applications in Developing Countries, Macmillan Press, London, 1987; Robert Schware, "The World Software Industry and Software Engineering: Opportunities and Constraints for Newly Industrialized Economies", World Bank Technical Paper No. 104, The World Bank, Washington D.C., 1989.
2. For example, the Indian Railway system has been totally transformed as a result of computerization, and the software and systems development was carried out by a local firm. Reservations, ticketing and scheduling has vastly improved now for rail travel in India.
3. Furthermore, these "import substitution" approaches have been critized as actually being "import reproduction" strategies, and thereby not contributing to domestic industrial development of a balanced and long term nature. See Lynn Mytelka, "The Unfulfilled Promise of African Industrialization", African Studies Review, Vol. 32, No. 3, pp. 77-137, 1989 for a critique of the traditional import substitution models.
4. See UNIDO, Industry and Development: Global Report 1989/1990, Vienna, Austria, 1990, for detailed data on productivity levels in individual countries and sectors.
5. Howard Pack, "Productivity and Industrial Development in Sub-Saharan Africa", Working Paper No. 3, Technology Assessment Policy Analysis Project, USAID, Washington D.C., 1990.
6. For example, Atul Wad, "Technological capabilities and Organizational Capabilities in Developing Countries", Working Paper, The Technology Assessment Policy Analysis Project, USAID, Washington D.C. 1990; Kurt Hoffman, "Technological Advance and Organizational Innovation in the Engineering Industry", World Bank Industry Series Paper, No. 4, March 1989, and Carlota Perez, "Microelectronics, Long waves and World Structural Change: New perspectives for Developing Countries," The World Bank, Strategic planning Review, Discussion paper No. 4, December 1989.
7. See, for example, Steven Hunt, Atul Wad and Timothy Lavengood, "Technology Assets Management: Optimizing the Returns from Technological Assets", Joint Working Paper of Arthur Andersen & Co. and the Center for the Interdisciplinary Study of Science and Technology (CISST), Northwestern University, 1991.
8. Perez, op. cit.

9. See, for example, Steven Hunt, Atul Wad and Timothy Lavengood, "Optimizing the Returns from Technological Assets", joint Working paper of Arthur Andersen & Co. and Northwestern University International Business Development. The strategic management literature also contains an increasing number of articles that deal with this issue. For example, David Teece, R. Jaikumar and C.K. Prahalad.
10. Drawn from Schware, 1990, op. cit.
11. Nicholas Damachi and H. Ray Souder, "Computers and Developing Nations", in Damachi et. al. (eds.) op. cit.
12. Hans-Jochen Schneider, "Software Production: Organization and Modalities", UNIDO, Vienna, IPCT.63, 17 May 1988.
13. Schneider, op. cit. p. 23.
14. See, for example, Schware, op. cit. 1989, which deals explicitly with the implications for the NIEs.
15. R. Narasimhan, "Guidelines for Software Development", UNIDO, Vienna, Working Paper, UNIDO/IS.439, 10 February 1984.
16. Schware, 1989, op. cit.
17. See the HASTE system, developed by ERT in Concord, Massachussets, which is a computer based system for predicting and mitigating potential impacts of a toxic chemical release. The system has already been installed in a number of chemical plants around the United States.
18. Lawrence W. Rosenfield, "Using Knowledge-Based Engineering", Production, November 1989, pp. 74-76.
19. See, for example, Richard Nelson and Sydney Winter, An Evolutionary Theory of Economic Change, The Belknap Press of Harvard University Press, 1982; Giovanni Dosi, ed., Technological Change and Economic Theory, Pinter Publishing, 1988; Jorge Katz, "Late Industrialization, Innovation Processes and the Theory of Technological Change: Notes Emerging from the Latin American Experience," mimeo, 1990 (notes for a contribution to Science, Technology and Development: A Sourcebook, Francisco Sagasti and Jean-Jacques Salomon, eds., to be published in 1991; and Sanjaya Lall, Building Industrial Competitiveness in Developing Countries, OECD, Paris, 1990.
20. "Transfer and Development of Technology in the Least Developed Countries: An Assessment of Major Policy Issues", United Nations Conference on Trade and Development (UNCTAD), Geneva, 17 August 1990.
21. Jeffrey James and Ajit Bhalla, "Micro-electronics, Flexible Specialization and Small Scale Industrialization in the Third World", Working Paper No. WEP 2-22/WP. 220, International Labour Office, Geneva, August 1991. The study they cite is by Raphael Kaplinsky, "A Case Study of Industrial Restructuring: From Mass Production to Flexible Specialization", IDS, Sussex, 1990.

22. See, for example, Michael Radnor and Timothy Lavengood, "Technology Intelligence and Sourcing: Issues and Opportunities for Developing Nations", Working paper, Center for the Interdisciplinary Study of Science and Technology (CISST), Northwestern University, 1990 and "The Technology Gateway Organization: A Mechanism for the Promotion of Technological Development and Industrial Competitiveness for Developing Countries", Atul Wad, Working paper No. 8, Technology Assessment Policy Analysis Project, USAID, December 1990.
23. Interview with Arun Tolani, President, ICIM U.S.A and formerly Vice-President, ICIM India.
24. See, for example, S. Soltysinski, "Strengthening Negotiating Capabilities in the Acquisition of Hardware and Software in Latin America", UNIDO IPCT.15, February 1987, Vienna.
25. Gillian Marcelle, "Industry Profile of the Jamaican Computer Services Industry", Consortium Graduate School of Social Sciences and Institute of Social and Economic Research, University of the West Indies, Kingston, Jamaica, 1991.
26. Marcelle, op. cit. p. 24.

Emerging Issues in the Selection and Distribution of Public Domain Software for Developing Countries*

Antonio Jose J. Botelho**
Caren Addis***

Acronyms

BBS	Bulletin Boards
CSW	Commercial Software
email	Electronic Mail
FDSW	Freely distributable software
ftp	file transfer protocol
FSF	Free Software Foundation
FUSW	Free UNIX Software
PDSW	Public domain software
SHW	Shareware
UUCP	UNIX-to-UNIX Copy Program

Glossary

backbone computer	A major switching computer on a network.
electronic mail	Transmitting files from one computer user to another, usually via communication lines.

* UNIDO/IPCT.145(SPEC), November 1991

** Information Specialist, Center for International Studies, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States.

*** Information Specialist, Programme in Science, Technology and Society, Massachusetts Institute of Technology, Cambridge, Massachusetts, United States.

Acknowledgements
The assistance of many professionals and users is gratefully acknowledged. Simson Garfinkel at MIT; Richard Salz of Bolt, Beranek & Newman (BBN); Richard Stallman and Robert Chasell of the Free Software Foundation; Paul Mayer, Bruce Robey and Randy MacLean of the Association of Shareware Professionals; and the many network users who responded to the questionnaire, offered invaluable suggestions as well as insights into the complex world of public domain and freely distributed software. The responsibility for the contents of this report lies solely with its authors.

file transfer protocol	A protocol that allows a computer user to transfer a file from one computer to another.
gateway	A computer which passes information between machines on different networks
host	A central network computer accessible by other computers on the network.
network	A collection of computers connected in such a way as to allow information to be passed easily from machine to machine.
site	A geographic location where computers are located within an organization. Also often refers to the host computer itself.
TCP/IP	A pair of networking protocols usually used together. TCP is the Transmission Control Protocol and is a standard protocol for transferring information from one computer to another. IP is the Internet Protocol and is a standard protocol for managing connections between one computer and another.
UUCP	A software communications protocol as well as a set of communications programs.
UNIX	A portable multi-user operating system.

1. Introduction

Over the past decade, as a result of the rapid decline in cost and increase in performance of microprocessors, and the consequent diffusion of microcomputers, the importance of computers in the realization of economic activities has continued to increase. As computers become more widely diffused and penetrate new economic and social activities, people have come to realize the crucial importance of software both for the efficient and effective utilization of computers. However, as computer hardware becomes more of a commodity, the software market has acquired greater strategic importance for both firms and countries.[1] This trend is leading to higher value being placed on software, as firms seek to acquire greater competitive advantage and higher profit margins through software; and as industrialized countries promote stricter control of issues of intellectual property and copyright and launch programs to turn communication and computer networks into strategic national assets. It is against this background that the importance of public domain software for developing nations has to be assessed.

An informal e-mail survey was conducted as part of this project among network users in several industrialized nations and a few newly-industrializing countries. The results reveal that among the main reasons for utilizing freely available software are its quality, cost, availability, functionality and easy of use. Technically-proficient users also singled out access to source code as an important reason. The major pitfalls are identification of right program, uneven quality of maintenance, reliability, and minor bugs. Users in NICs and western Europe singled out network connectivity as a major problem both for figuring out what is out there as well as for speedily retrieving programs.

This report examines the characteristics and dynamics of public domain and other low-cost software in the industrialized countries as well as their organization and distribution points. The objective is to propose a set of methodological guidelines for establishing an inventory of public domain software available in industrialized countries as well as to suggest the structure and operation of a clearing house for making the software available to developing countries.

The conclusion is that the decentralized organization and distribution of the software in the United States [and to a lesser extent western Europe] contributes positively to the dynamic of the markets — more specifically, the update, debugging, and customization of publicly available software. Although the distribution is very decentralized, there are key points in the system that act as information clearing houses.

2. Suggestions and Recommendations

Based on the experience of industrialized nations with the selection and diffusion of public domain soft ware and in the light of the perceived needs and of the technological context of the large majority of developing countries, which either lack affordable high-speed communication and computer networks, our suggestions for transmitting public domain and inexpensive software to the third world and our recommendations for future research are:

1. That key distribution points in the industrialized countries establish direct contact with distribution points or representative clearinghouses in the developing nations with as little intermediate intervention as possible. It is hoped that such a system will also lower administrative costs and will reduce the amount of "noise" in the communication between the developing country final user and the software provider.

2. The clearinghouses would preferably be regionally-based, would make use of existing local computer networks and international links, and would be preferably staffed by local software experts in close contact with industry, government at different levels, and academic and educational representatives from a broad array of economic and social activities, so as to insure that the local software needs of industry and society are well defined and provided for.

3. Communication among end-users should be strongly encouraged and facilitated. If possible directly, or as second-best option, through the diffusion by the clearinghouse of open, non-mandatory, lists and directories of users of software performing similar functions. These periodically updated directories could have users' evaluations of programs' strengths, weaknesses and major problems. The clearinghouse experts could relay doubts and contributions to user groups and bulletin boards in industrialized countries networks. This arrangement would make distributing and updating the software more efficient and effective, at the same time it would create an internal learning effect among end-users in developing countries. Incentives for the formation of vertical and problem-oriented user groups should be given.

4. The actual distribution of public domain software to developing countries' final users should take place primarily through the mail and in some cases, when widely available and costs are low, through national computer network systems such as bulletin boards or national software libraries.

5. In light of the widespread interest on the part of international and regional agencies on issues of diffusion and use of computer databases and networks in developing countries there already exist a number of sector-specific networks (academic R&D, health, space, physics, communications). In a few cases, these virtual networks have begun experimenting with the distribution of public domain software. There is therefore a need to coordinate and learn from these experiences as they develop, assessing their advantages and shortcomings. It is suggested that in certain network-poor areas some of these pre-existing networks could be employed as trunk lines for regional clearinghouses at a lower cost than that of developing dedicated networks.

6. In addition to the diffusion of computers, efforts should be made to encourage the establishment of bottom-up regional and local networks. This can be achieved through the provision of basic gateway services and low cost trunk and distribution communication lines. At the next stage, efforts should be made to allow for regions in different countries with similar industrial structures and/or social needs be connected in order to enhance the diffusion of public domain software and the learning effect of modifying, and eventually developing software.

7. Identification of actual and potential patterns of software usage and diffusion bottlenecks in developing countries should be carried out through cross-national sectoral surveys as well as through the establishment of a few integrated national software diffusion maps. This information would contribute to the design of FDSW distribution strategies complementary to existing commercial channels.

8. Typically, academics and other R&D personnel are the first to acquire any experience with FDSW in developing countries through their participation in generic (Bitnet, Internet) or specialized networks. An in-depth assessment of their unique usage pattern could provide valuable insights for the extension of their experience to other societal groups in developing countries. The questionnaire as well as the developing countries network lists developed in the course of this report research could serve as the basis for future research efforts in this direction.

9. An assessment could be made of existing programs for the diffusion of FDSW in vertical markets important for developing countries such as the educational market and the Scholastech (page 193) experience. Because these experiences often faced material obstacles similar to those found in developing countries, important lessons can be extracted from them.

10. In the long run, a central database such as Archie (page 198), could be developed, incorporating elements from the initial pattern of FDSW distribution and usage in selected developing countries. The aim would be to provide a speedier and more customized answer to specific developing countries' software needs and to avoid the emergence of duplicate efforts within a same geographical region or country.

3. Structure of the Report

The first section of this report defines and compares the different kinds of free and low cost software available in industrialized countries and briefly discusses their scope and applicability in developing countries. Freely distributable software (FDSW) includes a variety of software that can be copied, distributed, and used by anyone. Public domain software (PDSW) is the most common FDSW and its use and distribution are virtually free of restrictions. However, there are other kinds of FDSW, often available free-of-charge or at nominal fees, whose use and distribution carry restrictions.

The second section of the report discusses distribution methods in the United States and western Europe, which have centered on computer networks such as distributed bulletin boards; user groups; university and government computer centre software libraries, mainly for the scientific and technical communities; and mail for the general PC users as well as those transmitting very large amounts of data.

The third section discusses the applicability of these distribution methods in developing countries, based on their performance in the industrialized countries and taking into account the diverse needs, economic contexts, and communications infrastructure of developing nations. The report suggests that in the near future the mail system will be the primary source for diffusing software data and information, while computer networks will be used for identifying and transmitting the software as well as follow-up work such as debugging, updating and customization. In the short to medium term, computer networks are more likely to be used by the academic, central government and large business communities.

A fourth section will discuss other related issues including the logistics issues pertaining to the use and transfer of inexpensive software to the third world — hardware, operating systems, language and cultural barriers, business and social practices, and a compensation system for small amounts of money involved in distributing software.

4. Towards a Typology of Public Domain Software

There are three basic types of software: commercial software, freely-distributed software (FDWS), and public domain software (PDSW). Commercial software (CSW) is software that has legal protection guaranteeing the rights of the author, the company, or the organization that sells it. CSW cannot be copied without permission and does not provide access to

the source code. It is typically much more expensive than the two other types. At the other end of the spectrum is what is generically called PDSW, which in fact encompasses several types. In its purest form, the cost of PDSW is generally limited to distribution expenses.

In light of the shadings from one type of software to the next it is more appropriate to generically define PDSW as freely distributed, inexpensive or free software. **Freely distributed software** (FDSW) is software that can be freely distributed by anyone, although its use may carry restrictions. The FDSW with no restrictions attached whatsoever is called **public domain software** (PDSW). Other types of FDSW carry different types of restriction, although these restrictions may often enhance rather than limit its distribution. Within this latter category, two types of software have become more predominant and organized, although there are still other types available. Two of the more common types of software include **shareware (SHW)** and **free UNIX software (FUSW).**

In recent years, PDSW moved further from its hobbyist and hacker roots, as the majority of smaller computers users are now businesses. Yet, most of the software in use today is CSW. In contrast, much of the PDSW written over the past few years has been the result of specific projects and it was written for a defined purpose. Since the source code of the most popular operating systems and basic utilities (spreadsheet; word processing, database) is generally not available, most of the PDSW is being written with an aim towards enhancing the ease of use and functionality of these programs and in areas where market demand is too fragmented to justify the development of commercial products. As the software market becomes more commercialized and standards begin to play a greater role, user-supported software authors increasingly rely on shareware-semi-commercial arrangements to diffuse their products. There are, however, important initiatives in the technical segment of UNIX-based systems toward less commercialized and more open software markets.

4(a). Public Domain Software

Public domain software (PDSW) belongs to the public domain which is defined as "the realm embracing property rights that belong to the community at large, are unprotected by copyright or patent, and are subject to appropriation by anyone."[2] In other words, PDSW has no restrictions attached to it and is generally free. Any fees charged usually reflect costs in copying and diffusion, i.e cost of diskette or tape.

There are many kinds of PDSW for virtually every type of hardware and operating system although not all programs are available on all hardware/operating systems. The most popular tend to have the largest number of PDSW available. Often, more-technically oriented operating systems appear to initially generate a greater number of language- and utility-related PDSW given the technical proficiency and greater communication among their users, who share information.

As a general rule, PDSW programs are utility programs. That is, programs for system-related tasks such as making an operating system run smoother or quicker, transfering files (XMODEM; ftp) to organize a directory, or to provide communications to a network (KERMIT; PROCOMM) rather than specific applications (word processing, spreadsheets, databases). Yet, there is task and function-oriented PDSW for systems with greater public appeal: TxT, a powerful word processing; and a Logo language, in which source codes are often available. Games are also a popular category of PDSW as are valuable add-ons to existing popular functional programs (Lotus 1-2-3, dBase, PostScript). Add-ons make the program easier to use, give it new capabilities, increase its speed, or permit changes to be made in its interface. These include templates and command files, for example to send printer commands.

PDSW can be copied and used by anyone for any purpose. Some people might take public domain software, package it, and sell it to unsuspecting users who did not know that the same program was available free of charge. While this is not illegal it is considered a moral breach by the software community. Other people might change or modify a public domain program and sell it as proprietary or commercial software, a risk that is enhanced in developing countries due to the lack of general user access to wider information about types of software. Yet, by changing and then selling the software, the producer is taking on a variety of responsibilities regarding quality, documentation and support, which some argue overcome the fact that PDSW is incorporated into the program.

In industrialized nations, particularly in the United States, PDSW is easily available through a variety of sources. These include bulletin board systems (BBSs), local computer clubs (New York Amateur Computer Club), non-profit organizations (SIG/M for CP/M; Macintosh Special Interest Group), software libraries, academic and governmental computer centres, computer societies and associations (e.g. Boston Computer Society; San Diego Computer Society), user groups (Phoenix, Arizona IBM-PC users; CBASIC; MS-DOS), interest-specific groups, computer retail stores, small companies, computer company-sponsored groups, user group's archives (uunet; Young Minds), public libraries, commercial services (CompuServe; The Source; The Well) and schools, to name a few of the most common outlets.

User-groups are organized along a multiplicity of categories — hardware-type, software-type/function, application-type, function-type, language-type, hobby- specific user groups.[3] Many of these put out electronic lists or publish catalogues of PDSW, which are updated periodically. Altogether there are over a few thousand different individual outlets from which PDSW can be obtained. The channels available in each source, the services provided, the quality of the documentation, the ease of obtaining, and the basic costs vary within a small range. Generally a program is available from these outlets with some kind of documentation. At one extreme, some companies might charge a small fee for access to their collection, but may also provide a free demo or sample disk. At the other extreme, academic computer centres offer PDSW that has to be retrieved electronically through computer

networks or communication channels about which little information is initially provided, making its identification and evaluation problematic for non-technical users. Some limited support (*e.g.* start up documentation) may be available from the author, a BBS, or other users, which may require a certain degree of effort on the part of the user in identifying the other right users or the appropriate conference group. As a general rule, however, there are no formal support systems.

Most PDSW emanates from people who have written programs for their own use and then believe that they may be useful to others. Rather than undergo the burdens related to commercially launching their programs, they put it in the public domain. Others write PDSW because they enjoy it. Some of these people are also motivated to fight against CSW that does not permit users to modify it. The bulk of PDSW, however, comes with little printed documentation. Often authors will include a telephone number and address for support, which may be free or available at a nominal cost.

While PDSW is generally identified on a bulletin board, a computer club list, a review in an interest-specific conference, or a reference in a network discussion, it is usually obtained in the form of diskettes or tapes through the mail, following prior identification. In the past, access to networks was limited to the possession of a costly modem and restricted by cost and quality of telecommunication lines. In recent years, a few academics and certain professional categories have gained increased access to multitudes of interconnected networks. Today, file transfer programs (ftp) allow users to directly access PDSW lists and retrieve programs, in compressed on uncompressed form, and directly from file archives and libraries, both public and private. User groups and other non-profit organizations may charge nominal fees for processing the requests. Commercial ventures may package and improve PDSW and sell either complete libraries, individual programs, customized programs, or even charge for time of access to their library over phone lines or networks. This latter type is often referred to also as shareware or freeware.

While PDSW may exhibit more problems than CSW at the outset, over the long run it might evolve into a product superior to its commercial counterpart. The major problems with PDSW stem from the fact that because they are not commercial products they may not have been as extensively tested. Therefore, a new PDSW program that has not been widely distributed and used is likely to have bugs and small program errors, *e.g.* hitting a key outside the program capabilities may disable it[4]. PDSW programs at times may not run on certain versions of operating systems or certain equipment, as they were tailored to run on a particular version or a specific piece of hardware. These problems, however are often counterbalanced by the diversity of support that can be obtained which in the long run make PDSW popular because they have been tested and modified by a much greater number of user than commercial equivalents, and thus exhibit superior quality and ease of use.[5] User groups and other organized sources check every PDSW submitted at least once, and will often write short reviews or make small initial improvements.

4(b). Shareware (SHW)

Shareware (SHW) refers to a low-cost or free-of-charge arrangement for distributing software. Under the shareware concept, software can be freely copied and passed along to others, or distributed through bulletin boards. SHW is generally written for personal computers of all brands (PCs), but is rarely written for Unix.[6] SHW is often described as "try before you buy." The author retains the copyright of his/her software but permits copies to be made so that anyone might try it. If a user likes the sw, he or she is expected to register with and pay the author a fee, which generally ranges from about US$ 20.00 to US$100.00. By registering with the author, the user will usually obtain more complete documentation, a degree of support, as well as a channel to obtain knowledge about solutions to bugs and, more importantly, updates.

The main advantages of the shareware system are that it allows the user to try the software before paying for it; gives the user an organized access to a much broader base of available programs, and is substantially less expensive than most similar commercial programs. While shareware is based on the honour system it is not free software.

SHW, as a general rule, does not include source codes (like most commercial software). As a result, it may be difficult for the user him/herself to iron out bugs in the software or customize it. In comparison to CSW, however, it is easier to contact the author. Because the SHW distribution system is more informal than that of CSW, bugs may be ironed out more quickly. Shareware can be compared to commercial software in the following manner. If SHW is bought from a vendor (which usually charges US$1.00 to US$ 6.00 per disk) or directly from the author (through a recommendation from the trade association, for example) the SHW may well be of equal or even better quality than its commercial counterpart. Some commercial vendors of shareware may also provide some support. This support could be as good as that offered by a commercial software package with the exception that the user is likely to talk directly with the author or someone who has worked on the SHW. SHW can also be freely copied and passed on to friends. While this would be illegal with commercial software, SHW producers encourage it as a way to divulge the software. It is essentially a free distribution network for the SHW producer. Of course, if a user likes the program, he or she should register with the author and pay the fee.

If SHW can match many of the benefits of commercial software, then why do the authors not set up commercial production and distribution? The most common reasons are that authors do not have the expertise, time or capital to set up commercial operation — overhead, marketing, distribution, and advertising are expensive. They do not sell their software to a commercial venture because they want to keep the rights over it and believe that they can make money through the SHW distribution system.

The most successful shareware typically has a mass appeal in terms of functionality, it is low cost, offers some sort of support (telephone or good documentation) and has been around for a while. The quality of SHW is attested to by the fact that in North America, as well as in Brazil, where it has been divulged only over the past couple of years, many users are businesses, large and small, and a variety of educational and social organizations.

SHW authors may offer additional regular support at an extra fee.

As shareware programs become very popular, their authors develop a distribution system closer to that of traditional commercial vendors. Thus, PC-File's latest version, a popular shareware database, is sold through software retail stores and, for a fee, will provide telephone support to new buyers.

4(c). Freely-Available UNIX-based Software

Freely-available UNIX software (FUSW) is principally designed for minicomputers and workstations, and increasingly for PC networks, local and regional. Most of the software distributed in Unix networks is of the type where the author retains the copyright, and the ability to control it.[7]

There are very limited FUSW programs based on the shareware production/distribution system[8]. FUSW programs are mainly computer system administration-level and utility programs.

Among the best-known and highest quality FUSW is that of the Free Software Foundation (FSF). Its founder, R. Stallman, encourages software from the foundation to be freely copied and distributed with source code. Furthermore, FSF software is "copylefted." This means that it can be changed, but it cannot be sold. Once it is incorporated into any software, that software cannot be copyrighted. While some argue that this may eventually hamper the widespread adoption of FSF software, FSF hopes that its quality and transparency (availability of source code) will make its use more widespread.[9] Stallman says that the "Free" of FSF refers to freedom, not price.

The FSF plans to develop software that can replace and hopefully improve upon existing UNIX software, thus allowing users to modify and improve the software, as well as write with greater ease, because of access to source code, tailored applications. Part of this task has been accomplished: a program editor (GNU EMACS), a compiler (GCC), as well as other utilities have already been written. The group is working on a database, spreadsheet, and word processing program, which should be done in the next few years[10]. Much of the success of these utility programs stems from the availability of the source code, which permits programmers from all over the world to improve and diffuse FST software. GNU's ultimate objective is to allow anyone to run a UNIX-compatible system free and have the source code as well.

Another source of FUSW is the UNIX-like software produced by Berkeley's Computer Systems Research Group (CSRG), which is arguably freer than that of FSF. CSRG's efforts have been directed at rewriting the source code for UNIX, still controlled by AT&T, and making it available without any restrictions whatsoever to individuals, companies and organizations, which in turn can modify and resell it without providing the source code to the final user.[11]

There are other sources of free UNIX software, which may hold varying types of restrictions.[12] These sources are available through a few hundred archives and the programs are extremely varied. As a general rule, UNIX users are connected to some kind of BBS, and corrections and tips regarding the use of the software flow much more quickly and freely than they do in the commercial realm. Because users of FUSW are more technically proficient than the average computer user, programs are more complex and the dialogue among users more sophisticated.

4(d). Courseware and Government-produced Software

Another type of inexpensive software is courseware, commercially produced educational software. Because the writing of courseware is often subsidized by foundations and other non-profit organizations, much of the courseware available is inexpensive or given to the public domain. Furthermore, because of its objectives documentation tends to be of high quality.

For example, a non-profit organization called Scholastech has been involved in public domain courseware for many years. As part of its mission, the organization develops an argument that PDSW and some shareware actually belongs in a democratic educational system. The organization has screened several PDSW programs and has developed and given away a number of others with the objective of stimulating student creativity. Scholastech also has application-oriented objectives. Because educational budgets are often limited, schools often purchase second-hand equipment or accept a variety of equipment donations, which leads to incompatible hardware. Thus one of Scholastech's objectives has been to port courseware into different systems.

Government-produced or supported software is usually available for free or for a minimum set-up charge. However, to obtain this type of software one must qualify under the specific rules and regulations of the government agency in question. For example, the Center for Disease Control (CDC) has a library of PDSW in the area of epidemiology accessible to collaborating institutions and researchers.

5. Distributing PDSW

There are a multitude of distribution networks for FUSW, generically referred to as PDSW, in North America where the system is most developed. As a general rule they are organized by types of users, hardware, and/or applications. This discussion, however, will be organized by generic-type of software, consistent with the descriptions given above. While there are PDSW sources in most European countries, their access tends to be restricted to more technical users.

5(a). Distributing Public Domain Software

Public domain software (PDSW) is easily available in the United States, Canada and Australia, and to a less extent in western Europe (particularly the United Kingdom), through user groups, public libraries, schools, universities, stores that sell computer hardware and software, catalogue houses, professional group networks, and bulletin board systems. Some of the most popular sources of PDSW are university computer centres. Other non-profit organizations and government research institutes maintain large or specialized PDSW libraries which are accessible through several academic and research networks as well as private services. For example, MIT's AI Lab maintains a library of PDSW for the area that is accessible even from Europe (**prep.ai.mit.edu**).

Another example is CERN, based in Switzerland, which maintains a library of PDSW programs for the high energy physics area (CERNLIB). There are over 450 packages offered in both source code and object code form, 80 per cent written in FORTRAN and the remainder in assembly code. As mentioned above, CERN, as many other organizations of its type, has a strict set of regulations for access to its PDSW library. Collaborating institutions and university physics departments in member States can receive the programs and documentation free-of-charge. Commercial enterprises in member States may obtain the software for a fee. Enterprises in non-member States or intending to use the software for military applications are not permitted access to the library. The programs cannot be redistributed and CERN retains the copyright.

Probably the major source of PDSW, in binary and source format, for microcomputers is the SIMTEL20., on MILNET.[13] SIMTEL20 is also accessible through an Internet bulletin board, Info-IBM PC, which is a forum for technical discussion of the IBM — and compatible — PCs. The library accepts donations of source codes as long as they do not carry any restrictions, i.e. no fees, contributions, licensing agreements, are required or requested.

Every major non-profit computer network also has at least one PDSW library at its management site. Often there will be several libraries distributed over several sites. The three main networks are: BITNET, Internet and DECNet Internet. Other, virtual networks, such as USENET and UUCP-based networks are discussed below.

BITNET (Because It's Time Network), the worldwide academic and research network connecting academic institutions and collaborating research institutions, maintains a PDSW library. BITNET includes the United States and Mexican constituencies, NetNorth (Canada) and EARN (European Academic Research Network) and is composed of about 500 member institutions, with 100 new members added each year. Together they form a logical network that employs the same protocols and routing mechanism. The network serves over 6,000 computers in 35 countries, including the larger NICs such as Argentina, Brazil, Chile, China (Taiwan Province), Colombia, Côte d'Ivoire, Egypt, Hong Kong, Mexico, the Republic of Korea, Saudi Arabia and Singapore.

BITNET sites are limited to communication through file transfer and email. Files from its archives can be retrieved directly through ftp or a dial-up connection. Gateways exist to all other major networks. The BINET Network Information Center (BITNIC) maintains PDSW archives. Information about its archives as well as some PDSW that is not in BITNET can be gleaned from over 1,000 discussion groups. PDSW files can be transferred on BITNET.

EARN, the associated European network, is already connected to Côte d'Ivoire and connections are pending to six Middle Eastern countries: Algeria, Cyprus, Jordan, Morocco, Syrian Arab Republic and Tunisia. Two other participating developing countries outside EARN's geographical area are India and Pakistan. The network supports over 100,000 users connected by more than 800 nodes. EARN provides PDSW programs to enhance EARN usage. It also provides access and file transfer to other PDSW libraries through a specialized server (TRICKLE) for microcomputers stored at SIMTEL20.

Internet is a worldwide network, which allows the transfer of email, file transfers between computers, and interactive logins between machines. Internet was from the very beginning a large structure connecting many smaller networks, a structure that persists today. Internet utilizes the TCP/IP transmission protocol. In fact, Internet today is a logical network connecting more than 2,000 networks. It includes wide-area networks, such as NSFNET, MILNET, CSNET and ESnet; midlevel and regional networks, such as the MRNet (Minnesota Regional Network) or THEnet (Texas Higher Education Network), NORDUnet and SURAnet; and campus and organization local area networks, such as UTnet (Texas at Austin Network). Geographically, Internet links over 100,000 computers spread in networks in North America (including Mexico), western Europe, Japan, New Zealand and Australia.

DECnet Internet is the name of a worldwide collection of autonomous but cooperatively managed, regional, national and international networks based on Digital Equipment Corporation's (DEC) Decnet protocols. It is an important network for scientific and technical communities. The centerpiece networks include the United States Space Physics Analysis Network (US-SPAN) and the European High Energy Physics Network (E-HEPnet), and numerous university, state, national and international networks, such as portions of the internordic network (NORDUnet) and portions of THEnet. Application services include

email and remote file access and transfer, as well as interactive terminal-to-terminal communication. The specialized sub-networks maintain PDSW libraries with restricted access to participating institutions.

Other important network sources for the distribution of PDSW are the USENET virtual network, the microcomputer network FIDONET and several other regional, metropolitan and professionally-focused networks throughout the United States and Canada. For example, USENET's PDSW libraries have specialized sections on different hardware and software types, including GNU.

In western European countries, various sites have PDSW libraries, often with programs from major global sources (such as USENET's comp.sources) and locally developed programs. One such site in the United Kingdom is at Lancaster University (LANCS.PDSOFT), on JANET, the United Kingdom Joint Academic Network. In Switzerland, the iam.unibe.cha has a good library of object-oriented programs, accessible through the Internet gateway to EARN.

UNESCO's Division of Software Development and Applications has also been developing PDSW, primarily for scientific and technical information systems. Several institutes in developing countries use UNESCO's CDS/ISIS, and information storage and retrieval system adapted to minicomputers and microcomputers.

A major source for obtaining PDSW, or at least more specialized information about it, are BBSs. There are hundreds of bulletin boards over the dozens of existing public, semi-public and private networks. Just about everything is discussed in these BBSs: from agricultural research, to biotechnology, to epidemiology, to social services, to computer science. There are BBS discussion or conference groups dedicated to specific types of hardware (IBM-PC; Amiga); operating system software (MS-DOS; UNIX); or languages (C; Assembler; Fortran). Some BBSs are organized geographically. For example, Argentine, Brazilian, Egyptian and Indian students and researchers abroad maintain discussions with their home country and expatriate counterparts. Not all BBSs offer direct software downloading capabilities, but a majority of the larger ones do. Even if a BBS does not offer PDSW directly, it is likely to provide information about new and interesting area-specific PDSW.

Regionally and locally, an important source of PDSW, particularly for IBM-PCs are user groups (Boston Computer Society is one of the largest). As a general rule, software from user's groups is sent through the mail, copied and passed around at meetings, or sent over BBSs through telephone lines and a modem connection.

5(b). Distributing Shareware

SHW is distributed principally through vendors, but also informally among friends and acquaintances. Conceivably, distribution could also take place through many of the same mechanisms as the PDSW. Vendors set up mail order businesses where they maintain up-to-date archives and send out copies of SHW for nominal fees.

There are about 500 vendors of SHW in the United States — they charge anything from US$1.00 to US$ 6.00 to copy and mail a disk, and may also sell manuals of the most popular programs (PC-Write; PC-File)[14]. The end-user is still responsible for registering with the author. Vendors do not provide follow-up.

The Association of Shareware Professionals is such an alternative source of SHW. It reviews the software to make sure that it is not trivial (defined as a program that could not quickly be created by a programmer); creates and ensures contact with the SHW author for follow-up support; and consistent with SHW's "try before you buy" philosophy, makes sure that the program is not "crippled", which means ensuring that the the program can be fully tested. This avoids situations where the author might provide only part of a large data base's capacity and require the user to register before receiving the rest of the software.

5(c). Sources of Freely Distributed UNIX-based Software

Bulletin board systems (BBSs) are an important part of exchanging information about, obtaining, and diffusing UNIX-based FUSW. The most popular BBSs and archives for FUSW are in USENET, an international distributed bulletin board and discussion network. USENET, a virtual network within Internet, has over one-half-million readers from 17 countries in all continents of the globe and routinely carries discussions on almost 400 topics, organized hierarchically into newsgroups, which send and process articles. In fact, USENET is a sort of multi-topic bulletin board which a user can access to contribute and retrieve information, including PDSW. Participants from other networks, such as BITNET or UUCP (through UUNET, for example) may also access USENET, but direct retrieval of PDSW files is often time consuming and erratic.

USENET's **alt.sources** and **comp.sources** archives contain thousands of FUSW programs. The FUSW programs in these archives, and more specialized ones such as **comp.sources.unix** are also available from a variety of sites in the United States and western Europe Some sites will mail tapes. Solutions to bugs and upgrades are posted in **comp.sources.bug** or are relayed in the newsgroups.

The only costs associated with USENET are the communication costs as the software required to install a gateway to USENET is also in the public domain. In countries other than the United States, access to USENET is often made through a central location, which pays for the long-haul communications link to a central node. These systems may have an average, per-site cost to join, as in Europe. Other popular links to USENET are Internet and commercial-900 number gateways.

Access to USENET archives may occur through a variety of nodes and gateways. One of the most popular is UUNET, a large and specialized site run by a commercial non-profit organization, which also has over 600 megabytes of PDSW available for direct access on tape. UUNET also provides an alternative access to Internet and UUCP mail through its

dedicated communications relay computer, uunet.uu.net. Transfer of PDSW files from USENET archives may be made either with the UUCP transport protocol or TCP/IP protocol.

There are still other BBS sources for FUSW. BITNET automatic retrieval service will soon be available and work is being done to make the archives available to the general public via anonymous UUCP. UUNET permits subscribers to access and retrieve files directly. Some sites will include other PDSW archives such as games, X windows and GNU. A few companies in the United States (Motorola; Pyramid Technology) maintain copies of the archives and offer limited access.

As mentioned above, the Free Software Foundation archives provide free GNU software, which can be accessed from any of the major academic and research networks, as well as commercial services. An interesting new network source for PDSW is Archie, an Internet Archive Server Listing Service. Recognizing the difficulty of finding appropriate PDSW in today's multitude of networks and other PDSW sources in different sites and under different systems and protocols, McGill's School of Computer Science set up Archie, a dedicated-database of PDSW with over 2,600 entries which allows direct ftp. The archive lists the name of the PDSW program, document or package followed by a short description. The database hopes to incorporate non-UNIX info on PDSW in the future.

6. Diffusing PD Software in Developing Countries

6(a). Selecting Software for Developing Countries

There are a few basic issues to consider when weighing the advantages and disadvantages of a particular type of software:

- Quality of software
- Ease of use
- Hardware compatibility
- Operating system compatibility
- Program language
- Auxiliary software requirements
- Size of the program
- Computer memory requirements
- Media availability (i.e. size of diskette)
- Price of the software
- Tutorial availability
- Access to source codes, which permits a user to modify and debug a program, and eventually provide updates
- Access to user support, which permits a user to contact someone who can help sort out problems related to the use of the program

— User group existence which often permits faster debugging and informal help with program operation
— Quality of manuals and other documentation
— Restrictions related to its use
— Restrictions related to its further distribution

The main issue when judging the applicability of any FDSW or PDSW is the available hardware and operating system. PDSW and SHW can operate on many types of PCs, while UNIX-based software works most effectively on workstations.

The best operating system is a hotly debated issue in the industrialized countries — UNIX-based systems are considered technically superior; however, PCs run by DOS continue to be used by most people. One estimate, from a shareware vendor, states that in industrialized countries only about 5 per cent of the users have access to a workstation and UNIX; approximately 80 to 85 per cent of users use IBM-compatible with DOS; approximately 8 to 10 per cent use Apple-based systems; and CP/M and other operating systems are used on the balance.[15] Similar figures are not available for developing countries.

The performance of software is intimately tied to the user capability of clearly defining his/her own needs and selecting the appropriate program to perform the task. From this perspective, it is crucial to put as much information at the user's disposal as possible, assuming that the average user in developing countries is not necessarily technically proficient in computers and software as his counterparts in the industrialized world or his academic and research counterparts in the country.

Governmental administrative and service provider users (health, education, transportation) and private sector organization users, at all levels and particularly those in touch with the citizen or final consumer, are quite capable of defining their needs better than any computer expert. It is therefore crucial in the selection process to get as much information about the available PDSW to them as possible. Given the limitations of computer networks and even telecommunications lines in the majority of the developing countries, this information should be made available in printed support such as catalogues, directories and specialized lists. Great attention should be taken in organizing these lists so that they are easy to use and facilitate the selection and access to the needed PDSW.

As it is done in software catalogues in industrialized nations, some listings of PDSW found in the network archives provide some information about the program. Copies of these lists could be made available to more technically-proficient organizations in developing countries. In general, however, the PDSW products listed should be arranged in indexes arranged alphabetically, according to applications, according to system compatibility, and a cross of system compatibility/applications. Each item entry would include title, sub-title, type (PDSW; SHW; FUSW, other) version number, release date, last upgrade date (which together with the previous data may help assess reliability of item), author(s), email, phone or address for contact, news groups/discussion groups/users' groups referenced, compatible hardware, microprocessor, operating system, required language(s), memory requirements,

auxiliary programs required, type of support, price of program (in case of SHW), price of documentation, cost of support (*e.g.* cost per call), descriptive annotation. Some of the major applications heading would include accounting, agriculture, business management, construction, architecture, desktop publishing, engineering and science, food and lodging services, general services, health services, social services, databases, insurance; inventory, purchasing and invoicing, library services, manufacturing, media, medical, personal computing, programming tools, real estate, spreadsheets, communications and word processing.

6(b). Distribution Issues

There are a few well-established user groups that accept institutional memberships. In the United States the principal ones are in Boston and California. Memberships between local sites in the third world and the user groups could be set up. The membership fees usually include extensive documentation of the society's activities, access to new software and timely updates and debugging information on PDSW. Alternatively, United Nations supported clearinghouses and/or regional software distribution centres could have one membership and then distribute the software to local sites in the third world. The former option, because it diminishes the middle-people, could promote better contact between the user groups in the industrialized countries and the end-user in the third world. Conceivably the United Nations could negotiate "group-institutional rates" for third world organizations that are educational or research oriented.

There is no question that shareware vendors would welcome the opportunity to sell diskettes of SHW to the third world and also serve as distribution points for third world software in the United States. There were some cases of this with the Soviet Union.

User groups and Shareware vendors involve exchanging software for relatively modest sums. The problem here is that the cost of exchanging the currency frequently outweighs the amounts of payment involved. Some sort of compensation system would be critical to getting low-cost and free-of-charge software to the third world.

Another barrier to the free flow of software is the language in which the program commands and documentation is written. One way is to facilitate links with software user groups/vendors in countries within industrialized and developing countries that speak the same language. Additionally, the United Nations could fund programs for translating PDSW in developing countries into languages other than English.

6(c). Electronic Transmission Issues

There are three fundamental means of transferring software data from one place to another: by air, by phone, or through the mail. Air, or satellite transmission, has proved very complicated and costly in the United States. High-capacity phone networks needed for effective distribution of PDSW are expensive and therefore will be even less widespread

and available in the third world, except for academic and research applications. The mail system is slow; however it is a good option for inexpensively distributing software and/or large amounts of data that would be costly and tie up phone lines.

6(c)i. Satellite Systems

Little is known about Stargate, an experiment to transmit UNIX software by cable. It used the transmission facilities of a cable company and end-users had to buy a decoder, estimated between US$ 200 — 500 to be able to receive and decode the messages.

There were many technical problems: the signal was not always strong and properly tuned; the decoders were not always reliable (in part because there are no standards for this equipment). Errors were frequent and it was time-consuming to retransmit data and tune the signal. Furthermore, satellite time can be expensive.

Many of these problems are likely to be even more complicated in most third world locations. Furthermore, cable television is not as common in developing countries and satellite dishes would have to be used thus increasing the amount of hardware and the expense to the end-user. The system appears to be too expensive, unproven and complicated to be feasible for the diffusion of PDSW in developing countries.

6(c)ii. Computer Networks —Bulletin Board Systems

Computer networks are an important source of PDSW and related information, although this is not their primary function. PDSW and other freely distributed software is discussed and exchanged typically as a result of individual contacts among researchers with similar interests rather than as an activity in itself.

Computer networks are commonly used by academic and research communities to exchange information, documents, and programs; access databases and send messages. The networks are organized internationally, regionally and functionally, and may or may not include private firms. Many of these networks are connected by email gateways. The large national systems frequently restrict communications to those of a non-commercial nature, thus prohibiting commercial exchanges that might occur in some regional networks.

Many of these networks, including those in the developing countries, received start-up funds and equipment from IBM and later other large computer manufacturers. In some cases, these large manufacturers continue to fund the venture. In most cases, however, the networks are funded by membership and user fees and possibly government funds.

With the recent establishment of CREN (Corporation for Research and Educational Network), which resulted from the fusion of the CSNET and BITNET networks, the main global networks have moved to become closely interconnected. CREN's daily activities are managed by a non-profit consortium called EDUCOM, based in Washington, D.C.. BITNET provides access and command retrieval of programs from PDSW libraries in sites at other major networks and meta-networks.

In Europe, the principal network is the European Academic Research Network (EARN). The EARN network will soon have operational links to Turkey and Egypt. EUnet is a cooperative computer network run by the European UNIX Users Group. In Canada, the principal computer network is NetNorth.

A similar pattern is likely to emerge in developing countries, at least regarding international computer network linkages. In Côte d'Ivoire, for example, the first EARN linkage in Africa, the site is to act as a channel for West Africa to computer networks in the industrialized countries.[16]

It does not appear feasible to set up a computer network exclusively for exchanging FDSW. The network, in most cases, has to be funded by members and users, and it is unlikely that users who are exclusively interested in FDSW can afford high fees. Furthermore, in the third world, many of these projects are funded by international organizations such as AID or the World Bank. These projects aim to transfer specific technologies, many of which are commercial. This does not preclude the use of the computer network for FDSW but this objective would take a back seat to the principal goals of the project.[17] Computer networks, as in the industrialized countries, are likely to prove an efficient source of FDSW because of the organization of the network along users with similar interests or functions. Existing networks, as in the industrialized countries, are likely to become one conduit for FDSW related to the specific goals of the computer network members rather than a general conduit.

Some of the larger and more advanced developing countries are currently in the process of modernizing and extending their academic and research computer networks. Mexico's CONACYT has began to implement a high-capacity satellite-based Mexican Academic Network (RAM), which will be connected to NSFNET. RAM will eventually incorporate into a REDMEX the two existing satellite-based Mexican networks centered at UNAM (Mexican National University) and ITESM (Monterrey Institute of Technology). Two other parallel network projects going on in Mexico are REDLAED, organized by the United Nations and the Columbus Project.

Brazil's National Research Network (RNP) is in the process of implementing an academic and research network, which will replace the collection of *ad-hoc* BITNET connections that currently exist. The RNP also plans to install a distributed software library, which will offer PDSW and flat tariff software in the areas of engineering, medicine, social sciences and education.

In related developments recently, the UNDP contracted a non-profit socially-oriented network, PeaceNet, to set up an inexpensive, yet cumbersome, UUCP-based academic and research network in Bolivia and Cuba.

There are also about 300 BBS on Latin America alone, some connected to their home country. These mailing lists offer an important channel for the diffusion of information about and transfer of PDSW among the community of expatriate Latin American students in the United States, Canada, western Europe and Japan. Some of these mailing lists are devoted to technical issues in computer sciences, but they also discuss general issues and exchange news and information.

There are over 100 BITNET nodes in Latin America, the large majority in academic institutions. Brazil and Mexico have the largest number of nodes. In contrast, Argentina has over 100 UUCP nodes in academic and research institutions, as well as private enterprises and government departments. UUCP is relatively inexpensive compared to other network transport protocols (*e.g.* TCP/IP). The problem with UUCP, however, is that its routing structure does not allow for either remote login or real-time file transfer, making it unsuitable for the distribution of PDSW over long distances.

6(c)iii. Mail—A "Virtual Network"

A "virtual network" or "pseudo-networks" is a fancy way of saying that people would mail tapes and floppies to each other. Although slower than a network, it is a low-cost means of getting the job done. Catalogue houses and user groups in the industrialized countries could send their catalogues to user groups /universities /schools /firms /libraries in the third world. The software could be purchased by the institution in the third world and then it could be copied by individual users. Support for maintenance would be slow, but not insurmountable. As the software spreads in the third world, users in the particular locale could get together and work out problems themselves, thus creating invaluable skills.

7. Evaluating a PDSW Distribution System for Developing Countries

The central goal in setting up a distribution system should be to take advantage of the existing networks in the industrialized countries and the main existing gateways in developing countries to these networks. While the different networks could be connected to a clearing house in the United Nations, a better solution would be to connect the clearing house directly to the locale in the third world.

Direct contact between the end-user in the third world and the distributors and producers could preserve much of the inter-user contact, which has been important to the dynamism of these software markets in industrialized countries.

While the mail is likely to be the most cost-effective way of getting the software to users in the third world, *vis-à-vis* a network, it does have some disadvantages. One disadvantage is that the user cannot ask for and quickly receive problem-solving help on specific issues. This, however, is not unsurmountable. Not too many users in the developing countries currently have sufficient expertise in installing and maintaining workstation software. Even in the industrialized countries, it is common to retain "problem-solvers" or consultants to

teach the system and customize it for the user. While consultants could handle the larger problems for a few, the smaller problems could be dealt with through an appeal on a local, regional, or international computer network.

A more serious drawback of relying on the mail system rather than a network system is that the user in the developing country will be unable to contribute his/her knowledge or participate in the international hackers' interactions. While his/her participation may be limited at the moment, it also represents lost opportunities for learning. Under the economic constraints, however, the mail system is likely to outweigh the more expensive computer network system.

8. Issues for Future Research

- — Type, diffusion and trends in operating systems (OS) in the third world: Is DOS most prevalent now? What will happen in the near future? Should the United Nations do anything regarding OS distribution?
- — Logistical issues, apparently small issues that can create large obstacles in developing countries. Two that have been briefly discussed above and could be further explored in the context of different national settings are: translation and compensating checks. The latter is important anytime a transaction is made. Even free software will entail very modest distribution fees to cover the costs of the medium as well as mailing. The former is particularly critical for applications-oriented programs, less so for utilities.
- — Upcoming patent legislation — threatens all software, but certainly free distributed. A major potential issue with many recent threats. However, several factors will delay its impact. First, the international trade negotiations in services that have put intellectual property rights high up in their agenda appear to be stalled. Second, the courts in the United States have entangled the issue so much that it is unlikely that any final interpretations that could affect the production and diffusion of PDSW will emerge in the near future.

Notes

1. The United States Department of Commerce (1991) estimates that in 1990, sales of packaged software alone by United States software firms were $20 billion, or 40 per cent of the global market. Firms dependent on software revenues have pointed out that half of all software in use has been illegally copied. The Software Publisher's Association (SPA) has estimated that lost revenues derived from illegal software copying exceeds $2 billion a year.
2. Definition from Webster's Seventh New Collegiate Dictionary.
3. Estimates are that in North America alone there are over 3,000 user groups.

4. As a general rule, every software has some bugs in it, even commercial products, particularly when first introduced. Of course, larger and more complex programs will exhibit a greater number and variety of bugs, calling forth a more systematic maintenance.
5. S. Nutting, a well-known propagandist of PDSW says: *"The public domain is a collection of literally tens of thousands of hours of experimentation with the computer. Taken as a whole, it is a database of computer technique. The programs may or may not solve your problem, but by golly they will show you what that computer can do. It is a bubbling, burgeoning cauldron of uncontrolled experimentation, and it is absolutely marvellous."*
6. Inspite of much talk about opportunities for developing countries of UNIX as the standard of the future, the promise is still far from reality as today UNIX accounts for less than 15 per cent of the United States software market. For a discussion of the inflated promises of UNIX, see A. Botelho (1989).
7. It is suggested that the minimum serious investment needed to use UNIX-based software is a 386 co-processor and at least 150 megabyte hard disk. This, however, would only permit a limited degree of utilization of the UNIX-based software.
8. Shareware has not become an important avenue for distribution of workstation/ UNIX software, probably because not many average individuals outside the technical and academic spheres own workstations. As the price of workstations declines rapidly, there is talk that workstations will be tomorrow's PCs and UNIX the ultimate standard.
9. S.L. Garfinkel, "Programs to the People," Technology Review, February/March 1991: 53-60. See also, "Hacker's return," The Economist, 15 July 1989: 81-82.
10. GNU stands for a recursive acronym meaning GNU's Not UNIX. GNU is not UNIX but all GNU software can be run on UNIX. GNU EMACS has been adapted to a variety of computer systems, from supercomputers to desktop machines.
11. CSRG's most popular program is a UNIX networking software that has been incorporated into products sold by a variety of companies, including ICs.
12. D. Fiedler in "Free Software!" (Byte, June 1990), cites the example of where one author restricted the use of his UNIX software to non-military sites.
13. MILNET is the unclassified operational military network that resulted from the dissolution of ARPANET in the late 1980s.
14. PC-Write, for example, has about 50,000 people registered out of 100,000 who purchased diskettes, and has sold over 90,000 manuals.
15. The President of the Shareware Producers Association (United States) states that SHW has been unjustly accused of spreading viruses. There are no statistics on this, but there is no reason that SHW should be anymore responsible than disgruntled employees in commercial ventures or malevolent hackers over networks.
16. Efforts in this direction can learn from programs for regional cooperation in telecommunications, which is already being promoted with mixed success in two regions of Africa. See Rothery, Toure and Sharp (1989).
17. For a recent discussion see Bruce (1989).

Bibliography

Botelho, A. (1989) *Technological Change and Restructuring in the International Computer Industry: Perspectives and Policy Options for NIEs*, report prepared for the OECD Development Centre, Paris, mimeo.

Branscomb, A. (1988) "Who Owns Creativity?," *Technology Review*, May/June: 39-45.

Bruce, R. H. (1989) "Restructuring the Telecom Sector in Developing Countries: Some New Options for Policymakers," *IEEE Technology and Society Magazine*, 8 (4) December: 16-20.

Desbois, D. and G. Vidal (1988) "Abidjan devient le premier noeud Africain du réseau télématique," *Revue Tiers Monde*, XXIX (116): 1237-1243.

Dirschauer, S. (1990) "Free Software," *RUN*, November: 29-30.

Fiedler, D. (1990) "Prowling the Networks," *Byte*, May: 83-86+.

Fiedler, D. (1990) "Go Ahead, Make my Day," *Byte*, July: 81-84+.

Fiedler, D. (1990) "The Free Software Hit Parade," *Byte*, August: 85-88+.

Frey, D. and R. Adams (1990) *!%@:: A Directory of Electronic Mail Addressing and Networks*, Sebastopol, CA: O'Reilly & Associates.

Froehlich, R. A. (1986) *The IBM PC (and compatible) Free Software Catalog and Directory*, New York: dilithium Press.

Gassman,H.P.(1978) "DataNetworks: New Information Infrastructure," *OECD Observer*, October 1978: 10-16.

Glossbrenner, A. (1984) *How to Get Free Software*, New York: St. Martin's Press.

Hildebrandt, D. M., editor and compiler (1989) *Computing Information Directory*, Federal Way, WA: Pedaro, Inc. (6th edition).

Kahin, B. (1990) "The Software Patent Crisis," *Technology Review*, April: 53-58

LaQuey, T., editor, (1990) *The User's Directory of Computer Networks*, Bedford, MA: Digital Press

The League for Programming Freedom (February 24, 1991) *Against User Interface Copyright*, mimeo, 15 pp.

Lewis, L. T. (1989) "BITNET: A Tool for Communications Among Geographers," *The Professional Geographer*, 41(4) November:470-479.

National Research Council (Computer Science and Telecommunications Board) (1991), *Intellectual Property Issues in Software*, Washington, D.C.: National Academy Press.

O'Reilly, T. and G. Todino (1990) *Managing UUCP and Usenet*, Sebastopol, CA: O'Reilly & Associates (8th edition).

Rothery, R.; Toure, H., and Sharp, D. (1989) "Regional Cooperation in Telecommunications — Two Examples From Africa," *IEEE Technology and Society Magazine*, 8 (4) December: 11-15.

Samuelson, P. (1988) "Is Copyright Law Steering the Right Course?," *IEEE Software*, September: 78-86.

Sawusch, M. (1989) *Best of Shareware: IBM PC Utilities*, Windcrest/TAB.

Schaefermeyer, M. J. and E. H. Sewell, Jr. (1988) "Communicating by Electronic Mail," *American Behavioral Scientist*, 32(2) November/December: 112-123.

The Software Encyclopedia 1989 — System Compatibility Applications, New York: R.R. Bowker.

Todino, G. (1990) *Using UUCP and Usenet*, Sebastopol, CA: O'Reilly & Associates.

United States Department of Commerce (1991), "Computer equipment and software," *1990 United States Industrial Outlook*: 26-31.

Public Domain Software for Development*

Robert Schware**

The use of computers in government agencies and university and scientific research institutions in the industrialized nations has caused an enormous amount of software to be produced. Much software has also been prepared by United Nations specialized agencies, by multilateral and bilateral donor organizations, and by international foundations. A considerable amount of this software is in the public domain and has potential secondary applications in some industrial sectors. By "public domain" I mean non-ownership and this refers to software that is not classified or proprietary. For example, the United States National Aeronautics and Space Administration (NASA) offers more than 1,100 computer programs through its Computer Software Management and Information Center (COSMIC). COSMIC's inventory of computer programs that have been supplied to small businesses, universities and government agencies spans a wide range of application areas, such as computer graphics, circuit design and analysis, project management, energy system analysis, structural analysis, heat transfer and software development aids. Furthermore, a portion of the available public domain software helps solve the problems common to different government agencies and businesses and has been written in programming languages in such ways that minor modifications in requirements may be accommodated without too much programming effort.

This article is an attempt to provide developing countries with information on public domain software potentially applicable to their needs in order to minimize the re-development of programs already tested and in use elsewhere. (The term "developing countries" is used as a generalization since there are different levels of sophistication in the use and development of software within and among countries.) The article does not attempt to provide an exhaustive identification and description of all the personal computer-based models and software that could be of use to firms and/or institutions. Nor does it assess the relative strengths and weaknesses of the software available.

* *Microelectronics Monitor*, No.24, 1987/IV

** Senior Information Technology Specialist, Development Informatics Unit, Asia Technical Department, World Bank, Washington, D.C., United States.

The major benefits of obtaining public domain software for organizations and firms in developing countries are:

1. Such programs can broaden their access to a variety and number of software resources;
2. Overall costs, time and use of personnel resources for software acquisition and/or development may be reduced;
3. Computer program source codes may be obtained to study program capabilities and to modify or enhance such programs as needed; and
4. Alternative software choices can be compared before acquiring and/or developing software.

Despite these potential benefits, developing country firms and organizations may find serious problems in acquiring and modifying public domain software in the long term. Public domain software is a mixture of good programs and bad, old and new, useful and disappointing. Many good public domain software programs are poorly documented, which restricts their use to mostly experienced users who can quickly learn to use them correctly. Some programs come with no warranties or customer support. The quality of the software and the usefulness of the software output can vary considerably. Results from software and models may not easily be interpreted or may require interpretation and adjustment by so-called experts. Software may not easily be adaptable to a variety of countries. Many producers of modified software systems (in both industrialized and developing countries) have had to deal with large and costly changes in the functioning of programs. Some firms might not have sufficient resources to cover these costs and also become involved in the development of other activities, such as servicing and marketing. The basic point is that enthusiasm for acquiring and/or modifying existing public domain software should be tempered with some degree of realism.

At the outset, it is useful to provide a few definitions and to differentiate among types of programs available in the public domain. "Software" means all programs and routines used to extend the capabilities of computers, as distinct from "hardware" or "firmware". "Models" are software used either to explore broad, open-ended problems involving a significant degree of qualitative judgement (called "heuristic models"), or "algorithmic models" employed when factors central to the problem of concern are mostly quantitative and the relationships among factors can be expressed in arithmetic or algebraic form. "Available software" means software accessible to other countries (outside of the United States). In some instances, computer programs may be restricted to domestic distribution and use for a particular period of time. The cost of these public domain software programs range from nothing to several thousand United States dollars, though most often they may cost a nominal amount to cover diskette and document reproduction expenses. "Shareware" is a

copyrighted software for which the owner freely gives permission to copy, but with the request that the satisfied users send a contribution for maintenance of the program as well as to receive notices of updates.

Types of program

It is important to differentiate between two types of public domain software. The first generally has no formal distribution channels and is circulated among a surprisingly well-organized network of enthusiastic computer users through computer user groups and electronic bulletin boards. This type of software is part of a long tradition from the time of the first mainframe computers, when resourceful people wrote programs to solve particular problems and to show off their programming skills. Some sources for this type of software can be found in the "Other sources" section of this article.

The second type of public domain software — referred to in the following sections — is certainly less known than the former. It is developed and made available with funding from international development agencies, governmental agencies and research institutes. These programs are commonly prepared for submittal to agencies following set guidelines designed to encourage the development of a "software package" that is complete, well organized and "ready to use" in the public and private sectors. This means that the software package will basically consist of the complete program source code in machine readable form, supporting documentation that presents a full explanation of the program capabilities and the manner in which the program achieves its objectives and any additional information as merited such as sample input, output and data files. It also means that the software has been checked out and evaluated to verify the completeness and operation of the program, supporting documentation and suitability of the package for dissemination.

International Development Agencies

In the process of working to promote the social and economic development of their member countries, through loan operations and technical assistance, international financing and development institutions such as the Asian Development Bank, the Inter-American Development Bank (IADB), the World Bank, the Food and Agriculture Organization of the United Nations and the United Nations Industrial Development Organization (UNIDO) — to name just a few — have produced both general and sector-specific software and models that operate mostly on personal computers.

For instance, the Project Analysis Department of the IADB offers its members the following models, among others: "INDUSMOD", which generates financial projections for an industrial company or project over a ten-year period; "RURAL ROAD MODEL", which stimulates the economic costs and benefits associated with road projects providing access to rural areas or small villages; and "SPMOD", a financial projections model for public service institutions (communications, energy and sanitary engineering) for a period of ten years.

Many of these programs come in English and Spanish and operate in popular environments like LOTUS 1-2-3 and dBASE III. (For further information about these and other software packages in this article see Appendix I.)

Use of models and software in the broader context of planning and decision-making is receiving increased attention in the World Bank. One model for educational planning in developing countries — the "Economies in Curricula Choice (ECC) Model" — is used to test the effects of changing policies, *e.g.*, adding courses to curricula or increasing teacher salaries. This simulation model allows the user to study various alternative capital and recurrent costs scenarios created on the basis of a curriculum program, which can also serve to facilitate policy dialogue.

To meet the needs of highway authorities, particularly in developing countries, the Bank has developed a "Highway Design and Maintenance Model (HDM-III)" for evaluating policies, standards and programmes of road construction and maintenance. The model simulates total life-cycle conditions and costs and provides economic decision criteria for multiple road design and maintenance alternatives for individual road links or for an entire network of paved or unpaved roads. The user of this model can search for the best alternative, by way of discounted total cost, rates of return, net present values or first-year benefits.

Evaluation and monitoring trade and industrial incentives has become an important concern of many government agencies in developing countries. "SINTIA, Software for Industrial, Trade and Incentive Analysis" is a package designed to help analyze patterns of customs duties and quantitative restrictions in a country. It can be used to provide a systematic description of the nominal protection resulting from official tariffs and other import duties. It can also be used to simulate different nominal protection structures by allowing users to make different assumptions about reform scenarios (i.e., tariff changes with or without a devaluation), import elasticities and/or the effects of quantitative restrictions on price changes.

Another World Bank developed software package, called "WHAZAN", is a program for hazard assessment of industrial facilities to prevent future Bhopal-type accidents. A package of ten programs for computer-aided planning and design of cost-effective water supply and waste disposal systems has been prepared jointly with the United Nations Development Programme. The programs help designers and planners identify least-cost solutions and to examine cost implications of alternative designs performing a variety of tasks including the design of piped water distribution networks, the design of sewage collection systems, statistical analysis, mathematical optimization and financial screening.

UNIDO has developed two software packages for investment promotion applications. "COMFAR, the Computer Model for Feasibility Analysis and Reporting", is personal computer-based and designed for pre-investment studies or contract negotiations. "PROPSPIN, Project Profile Screening and Pre-Appraisal Information System", is the second package and is designed to speed the flow of industrial investment funds into developing countries. PROPSPIN reports are available in English and French.

Microcomputers and "appropriate" software are beginning to be seen in primary health-care programmes in developing countries. Some of the software is applicable to care providers at the "front line" of primary health care (PHC) programs — such as the Pan American Health Organization's "Primary Eye Care Consultation Program", which provides consultation to health workers on primary treatment of problems of the eye. Other software can assist managers at the district or central levels of PHC, such as the Aga Khan Foundation's program emphasizing integrated maternal and child health services.

United States Federal Agencies

The National Technical Information Service (NTIS) in the United States Department of Commerce is responsible for all functions relating to the acquisition, development and marketing of computer products. These responsibilities include identifying and acquiring machine-readable data files created for the Government, creating customized packages of information and of course providing public access to more than 1,500 databases and over 1,600 software programs. The programs cover a wide variety of subject areas such as energy, transportation, environmental pollution and control, industrial and mechanical engineering, biological and medical sciences and cartography. The NTIS Software Center collection includes programs from the National Energy Software Center, which is the United States Department of Energy's software exchange and information centre.

As mentioned above, the computer programs developed for NASA projects are distributed by COSMIC. Source code is pro ·ded for each program, so program capabilities can be studied and modifications or enhancements made. Program documentation is also available separately for reviewing capabilities in detail. The documentation includes user instructions and a detailed description of the equations solved and the techniques used to solve them. One program, the "Standard Assembly-Line Manufacturing Industry Simulation (SAMIS)" program, was originally developed to model a hypothetical United States industry, which manufactures silicon solar modules for use in electricity generation. The SAMIS program has now been generalized to the extent that it should be useful for simulating many different production-line manufacturing industries and companies. Programs distributed through COSMIC are usually restricted to domestic distribution and use for a period of at least one year. Some programs with implied military or strategic applications and industrial applications in advanced or highly competitive fields can be restricted to domestic use for longer periods.

Other Sources

A computerized project analysis program "COMPRAN" has been developed by the East-West Center and the Ohio State University to aid planners in assessing and comparing projects. The program offers financial and economic analysis capabilities, decision criteria options such as cost effectiveness, benefit-cost ratios, internal rate of return and sensitivity analyses, such as inflation rates, scaling factors on project costs and benefits. Its strength as a planning tool lies in its ability to incorporate monetized social welfare impacts into the analysis — a perspective often ignored in financial analysis programs.

A low-cost software project is under way at the Social Development Center to provide organizations in developing countries with a complete set of software with good documentation. The project is reviewing all of the available public domain and shareware programs available from electronic bulletin boards and specialized distributors and choosing one best program to be part of four sets of packages for office use, research, statistics and utilities (including small calculations, file keeping, writing notes, scheduling appointments, etc.). For example, the office package will contain the following generic applications: word processor, spreadsheet, graphics, database manager, form generator and key redefinition. Translations into Spanish and French will be available.

Related Literature

There is so much public domain software for virtually every brand of personal computer manufactured that it is often difficult to know where to begin a search for specific items. This last section provides current sources of published information, which for the most part contain public domain programs.

Managing a nation: The software source book is a review of software for application to a wide range of national administration and management activities. The book includes software and models addressed to issues of concern at a ministerial level that are relatively long-term in perspective, policy-oriented and applicable to many countries. Multi-sector and global models, modeling languages, and software for rural development; energy; water; agriculture; forests; population; environment and ecology; transportation and security, are included.

The Guide to Software for Developing Countries has been prepared by IBM. Its selections are divided into four major fields: agriculture, economic and social; infrastructure and physical, and administration. An Advisory Board screened submissions of hundreds of programs in 1984 based on the utility of the program to support development projects, proven performance in one or more developing countries, and availability free of charge or at nominal cost.

The Computers in Relief and Development Newsletter provides information about software for disaster management, response and preparedness. Emergency management related software is available specifically for emergency planning, event management, resource management and administration of relief personnel.

This article barely scratches the surface of public domain software. It ends by mentioning two generally reliable distributors of catalogues and of software — Public Brand Software and Public Domain Software Interest Group — who each offer more than 1,000 programmes, including word processing, communications, graphics, spreadsheets, business accounting, math and statistics, programming languages, utilities and games. If all this seems too good to be true, the distributors hasten to add that they do not guarantee that any particular program follows sound business practices

Appendix 1. Contact Addresses

Current sources of information are provided in this section for the software packages and related literature mentioned above.

International Development Agencies

The IADB

Project Analysis Department, Support Services Unit, Inter-American Development Bank, 1300 New York Avenue, N.W., Washington, D.C. 20577, United States, for INDUSMOD, RURAL ROAD MODEL, SPMOD, and other available software packages.

The World Bank

Economic Development Institute, Education Division, The World Bank, 1818 H Street N.W., Washington, D.C. 20433, United States, for the ECC Model; Industry Development Division, Industry and Energy Department for the SINTIA package; Project Manager, UNDP Interregional Project INT/81/047, Water and Urban Development Technology Unit, for Microcomputer Programs for Improved Planning and Design of Water Supply and Waste Disposal Systems; Transportation Development Division for the HDM-III Model; and Technica International, Lynton House, 7/12 Tavistock Square, London WCIH 9LT, United Kingdom for WHAZAN.

UNIDO

Feasibility Studies Branch, Industrial Investment Division, Vienna International Centre, P.O. Box 300, A-1400 Vienna, Austria, for COMFAR and PROPSPIN.

Pan American Health Organization

Pan American Health Organization, Opthalmology Department, 525 23rd Street, N.W., Washington, D.C., 20037, United States, for the Primary Eye Care Consultation Program.

Aga Khan Foundation

P.O. Box 435, 1211 Geneva 6, Switzerland, for the Community Health Program. Also ask for the Report of the Workshop on Management Information Systems in Primary Health Care, which contains information about other public domain software for primary health care programs.

United States Federal Agencies

NTIS

United States Department of Commerce, National Technical Information Service, Database Services Division, 5825 Port Royal Road, Springfield, VA 22161, United States, for NTIS software collection.

COSMIC

COSMIC, The University of Georgia, Computer Learning Annex, Athens, GA 30602, United States.

Other Sources

The Ohio State University, Department of Agricultural Economics and Rural Sociology, Room 226, Agricultural Administration Bldg., 2120 Fyffe Road, Columbus, OH 43210 United States, for COMPRAN.

Social Development Centre, Microcomputers for Social Development, 1313 East 60th Street, Room 476, Chicago, IL 60637, United States, for the four sets of packages for office use, research, statistics and utilities.

Related Literature

Global Studies Center, 1611 N. Kent Street, Suite 600, Arlington, VA 22209, United States, for *Managing a Nation: The Software Sourcebook.*

Communications and External Programs Manager, IBM Area South, 190, Avenue Charles de Gaulle, 92523 Neuilly-sur-Seine, France, for *The Guide to Software for Developing Countries.*

Computers in Relief and Development, 106 Park Road, Loughborough, Leice., LE11, 2HH, United Kingdom, for *Newsletter*.

Public Brand Software, P.O. Box 51315, Indianapolis, IN 46251, United States; and/or Public Domain Software Interest Group, 410 E. Sahara, Las Vegas, NV 89104, United States, for catalogues of public domain software.

The Production of Intelligent Products in Developing Countries

Hermann Kopetz*

The technical developments in the field of microelectronics will lead to a new spectrum of manufacturing products which integrate a mechanical subsystem, the microelectronics technology and the required control software into coherent products of improved functionality and reliability. These "intelligent" products will play a dominant role in the markets of the future since they will replace most of the "conventional" products of related functionality produced in the past and at present. Although not visible from the outside, a substantial part in the production labour of these "intelligent" products will be related to the required design and software effort. The manufacture of these products will be accomplished in highly automated factories. After presenting the characteristics of intelligent products, this paper discusses the distinct stepsthat have to be taken in order to produce such an intelligent product. In the final section some policy actions are considered, which can be taken in developing countries in order to promote the production of intelligent products.

1. Introduction

The dramatic progress in the area of microelectronics technology not only impacts the "classical" electronics industry, such as computers, telecommunications, industrial control and consumer electronics, but opens up a new class of "intelligent products", which integrate a mechanical subsystem with a computer control subsystem into a compact unit fulfilling a specific user need. These intelligent products will enhance and replace the "conventional products" (i.e. those products without microelectronics control) of related functionality.

Recent developments in the microelectronics technology and the design of integrated software/hardware systems have opened these new opportunities for product development in the area of intelligent products. Ten years ago, the design and construction of a special purpose computer tailored to the characteristics of a given application was a major and expensive project. Today, with the availability of standard microprocessors, standard peripheral chips and high level VLSI design tools, the implementation of such a project has lost some of its difficulties. Tomorrow, at a time when the VLSI design technology will

* Professor of Computer Science, Technical University of Vienna, Austria.

have matured even further, the construction of an integrated hardware-software solution with application specific functionality will be an accepted practice in the area of real time control systems for the volume market.

This paper about the production of intelligent products in developing countries is organized into three major sections. First, the characteristics of intelligent products are discussed and the future significance of these products for the high-tech market considered. The next section, the main part of this paper, concentrates on the design of these products and evaluates the advantages and disadvantages of the different implementation alternatives. In the final section, policy actions and recommendations for developing countries related to the production of intelligent products are presented.

2. Intelligent Products

The significant decrease in the cost of microelectronic systems has led to a multitude of new embedded computer applications, which interface directly to a user population unfamiliar with computer technology. The resulting products are destined for a growing mass market with important economic implications. We will refer to these new products by the term "intelligent" product.

2.1 Product definition

Let us introduce the concept of an intelligent product by giving some concrete examples:

- An automatic scale with an integrated microcomputer to perform calibration, weighing, conversion and calculation of some consequent value, *e.g.* the price, of some merchandise.
- An industrial controller, including the control valve, the computer and the control software.
- A washing machine with a microcomputer for the optimal control of the washing cycle in order to minimize energy use, water and detergents.
- A traffic light, including sensors and a microcomputer in order to improve traffic control.

An intelligent product is an autonomous embedded system that performs a specified service for its users. It generally consists of a mechanical subsystem, some sensors and actuators, a control subsystem with the appropriate functionality and a user interface.

2.2 Properties of intelligent products

In the following section we will analyse some of the important characteristics of an intelligent product.

Focus on genuine user needs

The most important property of a good intelligent product is its focus on a genuine user need. The ultimate success of any product depends on the relevance and quality of service it can provide to its users. The first step in the planning for an intelligent product is thus the identification of the user needs this product is to satisfy.

Optimal resource utilization

Because of their inherent information processing capabilities, intelligent products can provide the intended service with a minimal use of physical resources, such as energy, water, etc.. In times, when physical resources become more and more limited, this property of intelligent products is also of increasing macroeconomic importance. In designing intelligent products care must be taken that the internal use of electrical energy by the electronic components is also minimized, *e.g.* by the use of low power integrated circuits or by the use of solar power. This latter technology, if viable, is of particular importance for developing countries since it helps reduce operating costs and avoids all problems associated with batteries and their appropriate disposal.

Minimization of the mechanical subsystem

The number and complexity of mechanical parts which are contained in an intelligent product is minimized. This helps to reduce manufacturing costs and increase the reliability of the product. Almost all control functions are carried out by the smart control system integrated within the intelligent product.

Functionality determined by software

The functionality of an intelligent product is determined by the integrated software (in its widest sense). The effort required to develop this software can be substantial and amount to a significant portion of the overall development cost of such a product. Normally this software will be contained, either in a read only memory of a microcomputer, or in the design of a logic network. The product is then mass produced and distributed over standard marketing channels. After the manufacture of the product any correction of a software error or change of the software is very difficult, if not impossible, to realize. The quality standards for the software which is integrated in such an intelligent product are thus extremely high.

Simple to operate

An intelligent product will be utilized by an untrained user population with no computer expertise. Every action required by the end user to operate such a product must be designed from the point of view of the user's need in relation to the overall system functionality and not from the "computer's" viewpoint. It is therefore necessary to design special easy to operate user interfaces, both in hardware and software. General purpose terminal interfaces are not suited for these applications. Ideally, the use of an intelligent product should be self explanatory and not require any training or reference to an operating manual.

Ability to communicate

Although most intelligent products can provide the specified service autonomously, it is often required to interconnect an intelligent product with some larger system. In the above example of the automatic scale, it might be requested to provide an interconnection to a cash register. The protocol controlling the data transfer should be simple and robust. Generally, an optimization of the speed of transmission is not required.

High dependability

The control system and mechanical subsystem of an intelligent product (i.e. the machine to be controlled) form an integrated functional unit, i.e. a product. The reliability of the control system must be optimized in order not to compromise the reliability of the total product. Such an optimization of hardware reliability requires minimization of the chip count. This is achievable only by an integrated software hardware design. From the point of view of dependability, a VLSI solution for the control system is the best alternative.

Integrated diagnostics

The maintenance of an intelligent product should not require any special skills or tools. Intelligent products incorporate their own self-diagnosis software and can be maintained by an untrained worker replacing standard modules. In many instances the product is designed for the best reliability possible and has to be discarded in case of internal failures.

Mass production

Successful intelligent products are designed for a mass market and consequently for mass production. The mechanical subsystem, as mentioned before, is as small as possible in order to simplify the manufacturing process. In many cases the intelligent product will be assembled by highly automated robots in order to maintain a uniform and high quality level and reduce manufacturing costs. The initial investment in such a production facility can be substantial.

2.3 Significance of intelligent products

As has been pointed out previously, a large portion of the development effort for an intelligent product is related to software development and the setting up of an automated production facility. From one point of view, the marketing of intelligent products can thus be seen as a form of software marketing. However, the distribution of software via intelligent products has a number of obvious advantages over the marketing of standard software packages.

While the manufacturing of intelligent products is highly automated and not labour intensive, the distribution systems for these products must be set up carefully. The three key areas of concern determining the success of an intelligent product are therefore product development, production set up and distribution.

Tangible product for the mass market

An intelligent product is a tangible product, which can be marketed on its own. Since the integrated software/hardware product is produced by a single source, there are no questions of responsibility in case of problems at the hardware-software interface. Thus the customer will have increased confidence in the product.

No special user skills required

The potential market for intelligent products is not constrained by the limited computer education of end users. On the contrary, the simple user interface of intelligent products will make such a product easier to use than comparable conventional products. There is no special know-how required on behalf of the customer to perform the integration of hardware and software at the user site, as is the case with the distribution of standard software products.

Know-how protection

The most significant investment during the implementation of a computer application is in the area of software development. If the software is integrated with the hardware then this investment can be better protected than if the software is sold separately. At the moment, the adequate protection of software products is still an open question.

3. The Production of Intelligent Products

In this section we will distinguish between three distinct phases in the production of an intelligent product: the design phase, the implementation phase and the manufacturing phase.

The design and implementation of an intelligent product is quite different from the conventional software design or manufacture of a new piece of electronic hardware. Hardware and software of an intelligent product have to be designed in close consideration of each other in an integrated fashion. It is important to distinguish clearly between the system design phase and the implementation phase. In the system design phase the requirements for the product have to be established and the specification of the product, including its user interface and the functionality and performance of the control system have to be developed. In the implementation phase the appropriate microelectronics technology for implementing the specified design has to be selected. A well defined baseline between these two phases avoids the duplication of effort, in case a new implementation technology is chosen.

If the different implementation alternatives are supported by an integrated design environment, the switchover from one alternative (*e.g.* software on standard microprocessor) to another (*e.g.* a part of the functions in gate arrays) can be realized without extra overheads. These new implementation options have been opened by achievements in the area of VLSI design technology in the last ten years.

Starting with the seminal work of Mead and Conway [Mea80] the development of VLSI design tools has reached a state where it is possible to design application specific VLSI chips of moderate complexity within a period of weeks. At the moment a considerable effort is underway to integrate these design tools with the classical software engineering environments. In the not too distant future, it will be possible to consider the design of a VLSI solution as one of a number of alternative implementation strategies for a given system functionality. The integrated software-hardware design of the future will start with a computer aided requirement specification. These requirements will be checked for completeness in relation to established standards and internal consistency. In the following phase the system functions and system architecture will be specified. Finally, an implementation strategy will be selected.

3.1 Product design phase

The product design phase for an intelligent product can be partitioned in the requirements specification and architecture design phase. At the end of the product design phase, the architecture of the new product, including the external interfaces are fully specified. However, no decision has been made yet in relation to the implementation technology of the control system.

3.1.1 Requirements specification

The first activity in the development of an intelligent product is a feasibility analysis. Since such a feasibility must be performed for any kind of investment it will not be discussed further in this context. The result of the feasibility analysis is a market analysis and a first level specification of the functionality, the design, manufacturing and marketing cost of the new product, presented in the form of a detailed cost benefit analysis for this project.

The requirements analysis takes the feasibility analysis as its starting point. It must investigate the following topics:

Market penetration Analysis of the key markets for the intended product and specification of the product characteristics (functionality, reliability, maintenance strategy, acceptable price) required in these markets.

Critical user needs Identification of the critical user needs in relation to the intended product. Description of the typical system user, its background experience, expectations and training requirements.

Required system functions A complete analysis of all required system functions, covering the mechanical subsystem as well as to the control subsystem. The system functions should be classified in necessary functions, important functions and comfort functions. This classification is needed for the subsequent architecture design phase.

Interfacing requirements	A detailed description of the requirements on all system interfaces which are given by the environment of the new system. The requirements on the man-machine interface are of particular importance, since ease of use by the intended user is a determining factor for the success of an intelligent product.
Competing products	Analysis of competing products, including a description of the environment in which these products operate. Evaluation of the strong points and the weak points of the competing products.
Safety requirements	Critical failure modes of the intended product. Analysis of the consequences of system failure to the user. Reliability and maintainability requirements of the new system, including a description of the maintenance strategy. Rules, regulations, policies and other critical aspects of the intended application.
Statement of all assumptions	Analysis of the project in respect to the criticality of these assumptions.

Once the systems requirements have been established, these requirements must be validated. There are four criteria for validating the requirements:

Consistency: Is there a conflict between some of the requirements ?

Completeness: Are there any functions which have not been considered? Are there any constraints which may have been overlooked?

Validity: Are the requirements sufficient to cover the critical user needs? What are the key requirements in the intended market and are they covered by the product specification?

Realism: Are the requirements realistic considering the given market environment? Is it possible to design a product of the required functionality for a price which will be accepted by the market?

3.1.2 Architecture Design

The architecture design is concerned with the process of going from the statement of the requirements to the development of the system architecture, i.e. the specification of the subsystems, their interfaces and their interaction. The result of the architecture design is a document called the architecture specification. This document contains a complete description of all subsystems, such as the mechanical subsystem, the control subsystem etc., including a detailed specification of all subsystem interfaces

As has been mentioned before, this paper is mainly concerned with the analysis of the control subsystem. Therefore we will focus our attention on the specification of this subsystem. We propose that the control system specification consists of three sections (describing the control system from three different viewpoints): (i) the (static) structure of the control system, (ii) the dynamic behaviour of the control system and (iii) the performance of the control system.

Structural description

The structural description is concerned with the partitioning of the system, i.e. identification of the different subsystems and a detailed description of the interfaces between these subsystems. Since, from our point of view the interfaces of the control subsystem to the rest of the architecture are of special significance, we will concentrate our discussion on this particular interface.

The interface description must discuss the external view of the logical and physical appearance of the interface. On the physical level, this comprises the electrical and mechanical outlay of the connections and the coding of the signals. At the logical level, all objects visible at the interface between the subsystems, including their attributes and relationships must be described. Futhermore those internal objects, which are visible from the interface, must also be described. The inputs used to create, update or change these objects have be identified together with their domains.

The structural properties of the interface can be expressed in a connectivity diagram. The connectivity diagram depicts the input and output ports of the control system, including the name, type and coding of the corresponding data elements.

Behavioural description

The behavioural description specifies the input/output behaviour at the interfaces. It contains a detailed specification of the stimulus for an action and the processing steps, which have to be executed as a response to a stimulus. An action is normally started if a predicate on some input values and/or the real time (the stimulus condition) changes to "true". This stimulus condition has to be specified in the behavioural description.

Given the level of abstraction and the granularity of operations, we can distinguish between subsystems with internal state and subsystems without internal state. If a subsystem contains no internal state, then the action itself, i.e. the behaviour can be described in the form of truth tables, decision tables or mathematical functions. In theory, these stateless subsystems can be implemented by logic networks without internal memory.

For subsystems with internal state, the outputs are a function of the current state and the current input. At a given level of behavioural description the internal state space (the registers) and the state transition functions have to be separated. State transition functions and output functions (which do take the state as input, but do not contain any internal state)

can be described in the same form as "stateless" subsystems The internal state space which is relevant at the given granularity of operations has to be specified The overall behaviour of subsystems with internal state can be represented in some form of state diagrams or state transition tables

Performance description

The performance description concentrates on the timing properties of the interface. The timing of signals on the specified signal lines, the maximum response time of computations, the minimum interval between successive computations, etc. have to be specified in the performance description.

Test strategy

The detailed procedures for the product's acceptance test are part of the design specification. These tests must cover the functionality of the control system as well as its performance and reliability.

3.1.3 Design Tools

The effectiveness of any design methodology can be significantly enhanced if it is supported by an appropriate set of software tools, i.e. a design environment. We distinguish between architecture design tools, analysis tools, implementation tools and management tools [Kop86]. The architecture design tools support the system analyst in the requirements specification phase and the architecture design phase of a real time application development. The analysis tools can be used for an analysis of a given architecture design, *e.g* in respect to timing and reliability and a comparison of different designs. The management tools support the project management and documentation.

In present industrial practice, development teams for alternative implementation technologies *(e.g.* implementation by a microprocessor with the appropriate application software or implementation by an application specific VLSI chip) use different sets of specification, design and analysis tools. However, considerable research is in progress to integrate these different design tools and design databases into a coherent toolset so that the duplication of effort in implementing the same functionality with different implementation technologies is eliminated. However, such an integration can only be successful if the design engineers are experienced in both software and hardware design.

3.2 Implementation phase

Given the control system specification there is a wide spectrum of different implementation alternatives for the control system of an intelligent product. At one end we can select an off-the-shelf microcomputer and at the other end a custom made VLSI circuit. In this section we will analyse four implementation alternatives. a microcomputer implementation, a gate array implementation, a semicustom VLSI implementation and a discrete component implementation with MSI (Medium Scale Integration) packages.

The critical parameters determining the selection of the optimal implementation technology for the implementation of the control system of an intelligent product are task complexity, processing speed, reliability, power consumption and production volume.

Task complexity

Figures 1(a) and 1(b) show the qualitative dependence of the design effort and the marginal manufacturing cost for the mentioned technologies as a function of the task complexity.

From these figures it is evident that the discrete MSI component implementation requires the lowest effort for very simple products. However, the design and production effort increase sharply as the complexity increases. Since, by definition, the control system of an intelligent product is not very simple, this implementation technology will not be discussed any further in this report.

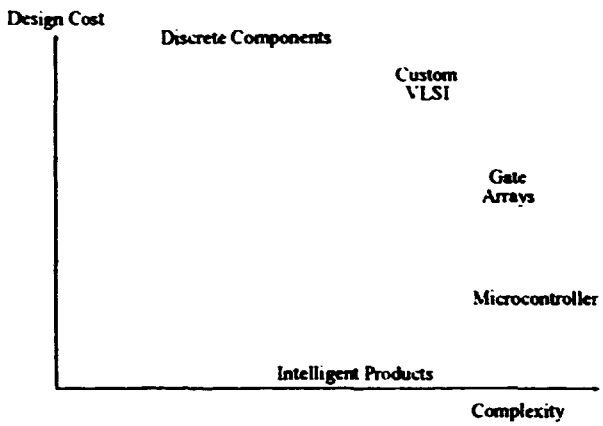


Fig. 1(a) Design Cost as a function of task complexity

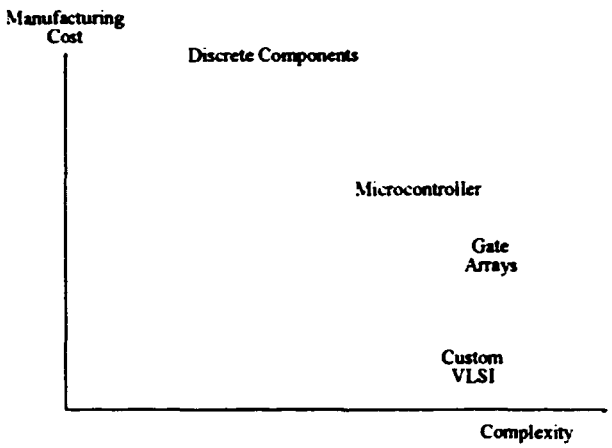


Fig. 1(b): Manufacturing cost as a function of task complexity

Processing speed

Tasks are realized by software (firmware) in the case of a microcontroller, but by logic networks in the case of gate arrays and custom made VLSI chips. It is evident that a logic network with a gate delay of a few nanoseconds is orders of magnitude faster than a microcontroller, where the execution of a single instructions takes about a a microsecond. Considering the complexity of even dedicated and simple operating systems, the response time of a microcontroller application will at best be in the order of milliseconds.

A full custom VLSI implementation has the highest performance and reliability (the smallest chip size), but requires the highest implementation effort. On the other end, the off-the-shelf microcomputer requires the smallest implementation effort (with the highest flexibility concerning changes), but gives us the lowest performance. The speed of a gate array implementation is much faster than the speed of a microcomputer, but slower than that of a custom VLSI implementation.

Reliability

In a first approximation, hardware reliability is a function of the number of pins and connections of a system. The reliability of the control system is thus optimized if the chip count and the number of connections are minimized. A full custom VLSI implementation dedicated to the intended control task gives us the best reliability. A single chip microcontroller solution with integrated read only memory (ROM), random access memory (RAM) and analog as well as digital inputs and outputs will also result in a good reliability of the end product. However, multiple chip implementations will require many extra pins and connections and thus reduce the overall mean time to failure.

From the point of view of implementation robustness, any system which minimizes the amount of internal state information is to be preferred to an implementation that contains a significant amount of internal states. If the same functionality is realized by a logic network versus a stored program implementation, the logic network will result in a more robust system.

Power Consumption

The power consumption of a dedicated VLSI implementation in a low power technology, such as CMOS, will be less than that of a functionally equivalent gate array or microcontroller implementation. It depends on the overall product characteristics, whether the power consumption is a critical parameter in the selection of the implementation technology.

Production volume

A qualitative comparison of the initial design effort and the marginal manufacturing cost for the different implementation technologies is given in Figure 2.

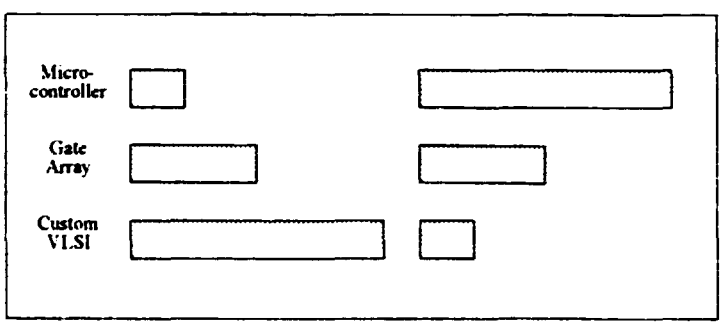


Fig. 2: Initial investment and marginal manufacturing costs for different technologies.

Criteria	Micro Computer	Gate Array	Semicustom VLSI
System development time	short	medium	long
Flexibility	high	medium	low
Processing speed	slow	high	highest
Chip size	large	medium	small
Power Consumption	high	low	very low
Know-how requirement	medium	high	highest
Initial investment	low	medium	high
Unit cost low volume	medium	high	highest
Unit cost high volume	medium	lower	lowest
Reliability	medium	higher	high

Table 1: Comparison of implementation technologies

It has to be pointed out that significant efficiencies are achieved if the implementation of the control system can be kept on one or, at most, a few chips. Going from one chip to another causes duplication of circuits as well as larger and more power consuming circuits, resulting in additional delays and cost. Furthermore, the negative effect of additional connections on the overall reliability of the system has to be considered.

Table 1 presents an overview of the characteristics of the different implementation technologies. In battery powered intelligent products, the expected power requirements can be of overriding concern. In these applications a low power (CMOS) application specific VLSI solution can be the preferred alternative. In other applications, where the system is to be connected to a power source, other considerations may be more important.

3.2.1 Microcomputer implementation

The most interesting alternative for the implementation of the control system of an intelligent product is a solution containing a complete computer including the required Input/Output interface on a single chip. Such a chip is called a microcontroller.

A microcontroller is an off-the-shelf VLSI chip with on chip CPU, RAM, ROM and process I/O. It can be considered as a standard "system on a chip". The specific functionality of a microcontroller is realized by application software stored in the ROM. Only the recent advances in VLSI technology have made it possible to economically produce chips of the complexity needed for microcontrollers. A typical state-of-the-art microcontroller has a powerful 16 bit CPU, which runs at about 10 Mhz speed, provides 8 kbyte ROM and 256 byte RAM memory on chip, has 64 digital input output lines and 8 analog input channels

and a serial communication link. In the operating mode it has a power dissipation of about 100 mW and in a standby mode, where the contents of memory are maintained, a power dissipation of 100 μW.

The software development for microcontrollers is very similar to standard software development. Many microcontrollers come in different versions for program development and production. The program development versions support RAM or EPROM memory for the storage of the program code so that the software can easily be modified. There are standard software development and debugging kits available from most microcontroller suppliers. In the production version the application software is contained in a ROM memory and cannot be modified any more. Extensive tests of the application software have to be performed before it is committed to ROM.

When the required performance can be achieved by an off-the-shelf microcontroller, the electrical power dissipation is not a problem and the production volume for a particular version of the product is not high (*e.g.* less than 10,000), then a microcontroller based implementation is the most economical alternative, both from the point of view of design and the point of view of production.

3.2.2 VLSI implementation

A custom VLSI implementation of the control system should only be carried out if it is not possible to find off-the-shelf components of the required functionality. Even if only less than half of the functionality of a well tested standard component is used, it might be more cost-effective than the design of a new custom chip.

If, for whatever reason, a microcontroller implementation of the control system is not feasible, then a VLSI solution must be considered. A VLSI chip designer has several choices, each of them offering trade-offs between chip density and chip design time. At one end is the standard gate array approach and at the other end the full custom design of a VLSI chip.

If the task of the control system is implemented in the form of a logic network in VLSI, the transformation of the behavioural specification into a logic network specification has to be performed. Although considerable research effort is devoted to this interesting topic, at the moment this transformation cannot be accomplished automatically. However, there are many CAD (Computer Aided Design) tools available today which support the engineer in the generation of the physical design representation (mask generation) once the logic network has been specified.

It can be expected that sometime in the future new development tools, called silicon compilers, will be available, which perform the translation of an application domain specific high level behavioural specification into the logic specification and further on into mask representation required for the production of the VLSI chip. A silicon compiler has to

contain a knowledge base, which understands the physics of the devices and all design rules of a technology so that the detailed geometrical mask shapes can be built up from the high level structural and behavioural description of the required system.

The implementation of the control system in the form of a logic network is the best way of protecting the application know-how, i.e. the software.

Gate array implementation

The gate array technology is based on the concept of having a fixed basic pattern of logic gates on an integrated circuit, which is then "programmed" by customized on chip wire connections in order to produce the logic network of the intended functionality. The main advantage of gate arrays over microcontroller implementations is a speed improvement of three orders of magnitude.

Gate arrays have become popular because of the fast turn-around time of the semiconductor foundries and the availability of powerful design and simulation tools. A complex gate array can now be designed and developed, and prototype samples produced in the same time frame as it takes for the production of a printed circuit board. On the other hand, every production run of a gate array results in significant costs for the customer. Therefore the customer must check his design very thoroughly by extensive tests and simulations before it is committed to production.

From the point of view of the semiconductor manufacturer, gate arrays are a mass product. Only a few customer masks are required at the end of the manufacturing process, all other masks being standard. Since the responsibility for the correctness of the customer masks is in the hands of the customer, the semiconductor manufacturer does not require skilled design engineers. The complexity of gate array based control system is increasing steadily. Systems with 10,000 and more gates are becoming quite common.

The main disadvantage of gate arrays is the large chip size because of the necessary spacing between every pair of adjacent columns of gates required for the wiring of the chip. As a rule of thumb, the gate array chip size will be about five times larger than the size of a custom designed logic network of the same functionality with all consequent effects in reliability, production cost and power dissipation.

Custom VLSI Implementation

The implementation of a given functionality by standard cell arrays is the next logical step for semi-custom component design. A standard cell is a pre-defined and pre-tested circuit, which performs the defined function. There are standard cells available for all kinds of digital functions (*e.g.* adders, registers, etc.) and analog functions. It is possible to combine digital and analog functions on the same chip in order to avoid the problems of inter-chip connections. Provided the available cell library is sufficient for the given task, the VLSI

designer has to place and interconnect the cells under the guidance of a CAD system. Normally, the CAD system will also contain facilities for simulation of the cells so that the design can be checked before it is committed to production.

In the fixed image standard cells [Dec86] a set of fixed sized cells is arranged into an array separated by wiring channels. Sometimes a pre-defined I/O path configuration and a power distribution system is provided by the design system. The next generalization is the variable image cell. If all variable image cells are of the same height, then a regular structure of the chip is achieved by interleaving an array of cells with an area for the wiring. The most general form is the full custom design. Such a design requires a very high engineering effort and should only be considered if a large mass production (in millions of chips) is foreseen.

3.2.3 Manufacture

The added value in the market for intelligent products is not in the manufacture of the microcontroller or the silicon chip, but in the product design, software development, assembly and marketing of the product. It is therefore expedient to buy the off-the-shelf microcomputers for the control system or to take advantage of the services provided by silicon foundries in many different countries.

If a particular intelligent product is successful and a mass market develops, the assembly work should be highly automated in order to guarantee a sustained high level of product quality and to stay competitive in the international market place. Only products with the highest quality attributes, which are manufactured at internationally competitive costs can compete on the world market.

4. Policy Actions

A high tech industry, such as the production of intelligent products, can only succeed if a conducive infrastructure is established and maintained by appropriate policy actions. In this section three areas, where policy actions by the public sector are necessary in order to support the growth of high tech industries such as the production of intelligent products, will be discussed.

4.1 Provision of training facilities

Looking at the speed of change in the field of information technology it is of paramount importance to realize the important long-term trends in their early phases in order to take appropriate actions in the educational system.

Only if engineers of the required background in software and hardware are trained in a given country will the industry be in a position to take advantage of these new trends. The most important long-term action relates to the establishment of courses of study on integrated software-hardware design techniques at trade schools and at the university level for interested students. These courses must be backed up by appropriate laboratory exercises.

Teaching software and hardware technology without the possibility of practical work on the machine is a dangerous undertaking. Since the lectures tend to become too theoretical, the student will not grasp the elementary concepts and might shy away instead of developing a positive attitude towards this new technology.

Therefore, any education initiative in this field must be supported by an initiative to provide the necessary computer equipment and software for the practical training and access to a silicon foundry.

4.2 Support for the start-up of small high-technology companies

In any society you can find talented people willing to take the risk and benefits of becoming an entrepreneur in the field of high technology. The policy in the field of technological development must provide an economic and political climate so that these young entrepreneurs will succeed in the founding and operation of new companies.

In many industrial countries the operation of new "high-tech industrial parks", where young entrepreneurs can find the organizational and legal support for the operation of a new company, is well established. Often these industrial parks are affiliated to a research laboratory or university to provide contacts and access to the technical infrastructure for product development. It is felt that a similar organization should be set up by the public sector in developing countries.

The start-up phase is a very critical period for a newly founded high-tech company. In this phase new products have to be designed and developed without any income from older products, which have been introduced on the market already and could contribute to the cash flow. In this limited phase public support through research and development contracts for prototype product development can be of significant importance to the financial viability of the newly founded company.

4.3 Marketing support

Normally, entrepreneurs are fascinated by the technical characteristics of their new product and do not pay sufficient attention to the development of the market. It is particularly important to support the new companies in this field. Established organizations such as the economic divisions of large banks or the import/export branches of trading houses can be (financially) encouraged to advise and cooperate with newly-founded small technological companies in this critical area of marketing. Also, the procurement policy of the public sector should favour newly founded high-tech companies formed within a country.

Summary of Recommendations

Let us summarize the recommendations for action by the public sector in this high-tech field as follows:

1. Establish a core group of experts with sound expertise in the following fields: software engineering, hardware design and implementation, application know-how, application software development, organizational, marketing and legal skills.

2. Establish a training programme for computer engineers and teachers at universities. Introduce courses on software development, VLSI design and process control into vocational training schools. All courses must have a substantial section of practical laboratory work. These training activities should be initiated by the expert team

3. Initiate a programme for the financial, organizational and legal support of small companies. Provide these small companies with research and development funds for the start-up phase.

4. Encourage existing organizations, *e.g.* banks or trading houses, to cooperate with newly founded small high-tech companies in the area of product marketing. Provide financial incentives for such a cooperation.

5. Identify an application area, which is at the centre of national priorities and coincides with a genuine user need and which can be effectively supported by some intelligent product. Provide funds for a research project in this area.

6. Initiate a pilot project in the selected application area with an intelligent product destined for the end user market. Involve the established expert group with the understanding that a new company is to be formed. Closely monitor the progress of this project. It is important to provide a good design and an excellent, i.e. very simple, user interface for this pilot product.

7. Ascertain that the acquisition of intelligent products by the government and other public or semi-public agencies is open to and favours these new local high-tech companies.

References

[Alf77] Alfor,d M.W., A requirement engineering methodology for real time processing requirements, IEEE Trans. Softw. Eng. Vol SE-3, p.60-69, 1977

[DeC86] DeCamp, W.F., Sporynski, G.A., Burbank, H.C., Gate Array and Standard Cell Approach in: Design Methodology, S. Goto, editor, Elsevier Science Publisher B.B., 1986

[Kop86] Kopetz, H., Design Principles for Fault Tolerant Real Time Systems. Proc. of the Nineteenth Annual Hawaii International Conference on System Science, Western Periodicals, North Hollywood, Cal., 1986, p.53-62

[Mea80] Mead, C., Conway, L., Introduction to VLSI Systems, Addison Wesley Publishing Company, Reading, Mass, 1980

[Was83] Wassermann, A.I., Freeman, P., ADA Methodologies, Concepts and Requirements, ACM Software Engineering Notes, Vol 8, Nr. 1, Jan. 1983 p.33-98

[Mur82] Muroga, S., VLSI System Design, When and How to Design Very-Large-Scale Integrated Circuits, John Wiley and Sons, New York ,1982

VI

Conclusion

CONCLUSION

In past years the UNIDO secretariat has promoted the concept of software as an industry and the actions that developing countries could take to promote that industry. The concept has been elaborated through several studies dealing with software development and applications for developing countries, the approach to software production in those countries and guidelines for organizing software houses. Further work in this area includes the promotion of software links to industry for specific applications of relevance to developing countries and the collection of papers presented here in the hope that these will contribute to this goal.

Recently UNIDO has concentrated on the following major trends in the software field, as is also reflected by the collection of papers presented in this publication:

(a) **Increasing demand, particularly for software, in the wake of the most recent developments in microelectronic chips**. The availability of reasonably priced and powerful microelectronic chips is creating an increasing demand for system and applications software based on those chips. Many urgent user needs could be satisfied most economically by taking advantage of these latest hardware developments. The demand is principally in the areas of distributed system software and applications software that supports easy-to-use man-machine interfaces and integrated hardware-software solutions;

(b) **Globalization of the software market.** The standardization of computer hardware brought about by mass-produced microelectronic chips and the availability of a responsive global communication infrastructure (computer networks such as Internet) has led to close transnational cooperation among software professionals. Information transfer achieved by these new means of communication ensures the global community comparable levels of know-how. Ultimately, a product either succeeds globally or disappears: even "niche" markets are being globalized;

(c) **Optimal quality of product, documentation and development processes**. As a direct consequence of the globalization of the software market, only products of the highest quality succeed. Quality refers to absence of design faults, ease of use and detailed documentation. It is also accepted that software quality is determined by a rigorous development process; thus, emphasis is now placed on structuring the development process accordingly. Many software users have already begun to demand full accreditation of the software development process;

(d) **Increased cost of entry into the software market**. The globalization of the software market and the emphasis on assured software quality have led to a significant increase in the cost of entering the software market as a newcomer. In particular, the cost of global marketing can be prohibitive for a small company. Schemes already promoted by UNIDO

for joint-venture cooperation and risk-sharing between companies in industrialized and developing countries are of immediate relevance in this context and should thus be encouraged further.

The key issue in the development of software products is a demand for computer applications. Country-specific strategies for software industry development, including software applications in non-electronic industries and services, such as cement, fertilizers, agriculture, chemicals and transportation, have to be developed to accommodate the mix of infrastructure, policy-framework and manpower available in each developing country.

In order to improve product quality and effectiveness in the small and medium-scale industry sector, particular emphasis is given to demonstration projects for that sector. Demonstration projects yielding tangible and measurable improvements in productivity in various sectors of the economy are of vital importance to opinion building among planners, economists and administrators in developing countries.

Developing countries recognize value-addition as a valid objective for the software industry. Contract programming, custom design and implementation would help build up a critical mass of experience. It is also important for developing countries to acquire expertise in leading-edge aspects of software technology.

Developing countries should also consider adopting a coherent and cooperative strategy in the area of software protection along the lines already adopted for integrated circuits.

A large proportion of applications, especially in the small and medium-scale industry sector, could be implemented using small computers. UNIDO efforts are directed at ensuring that industrial users in developing countries are fully aware.

There is a clearly perceived need for centres of excellence in software production and applications. UNIDO encourages and supports developing countries in setting up microprocessor and/or software applications development centres to address applications in domestic industry and to develop new, intelligent products. UNIDO also lends infrastructural and related support to developing countries in setting up software development parks where professional entrepreneurs can set up operations serving both domestic and export markets.

UNIDO works out methods for effectively interfacing the research results and practical industrial use. Both centres and networking could be organized so as to bring about the effective industrial application of research results.

The most critical input to software development is the quality of the human resource capabilities available. Consequently, in order to build up the software industry in developing countries, it would be necessary to build up human capabilities in the field of software technology in two specific groups:

(a) Training those about to enter the software development industry;

(b) Retraining and/or upgrading the skills of professionals already active in the industry.

Assistance is provided to programmes designed to ensure the cost-effectiveness and relevance of training programmes to industry. With respect to item (b), UNIDO supports countries in setting up programmes designed to acquire, analyze, adapt and disseminate relevant technical information to software developers, as well as to analyze the structural or organizational implications of informatics in both an industrial and administrative environment.

Emphasis is placed on the importance of supporting the growth of the software industry in developing countries and the pursuance of active policies to promote the diffusion of informatics, particularly among small and medium-scale enterprises.

The approach outlined here is expected tc assist developing countries in the inevitable transfer to the information-based economy. This book is also intended as a contribution in this direction.

UNIDO GENERAL STUDIES SERIES

The following publications are available in this series:

Title	*Symbol*	*Price ($US)*
Planning and Programming the Introduction of CAD/CAM Systems A reference guide for developing countries	ID/SER.O/1	25.00
Value Analysis in the Furniture Industry	ID/SER.O/2	7.00
Production Management for Small- and Medium-Scale Furniture Manufacturers A manual for developing countries	ID/SER.O/3	10.00
Documentation and Information Systems for Furniture and Joinery Plants A manual for developing countries	ID/SER.O/4	20.00
Low-cost Prefabricated Wooden Houses A manual for developing countries	ID/SER.O/5	6.00
Technical Criteria for the Selection of Woodworking Machines	ID/SER.O/11	25.00
Issues in the Commercialization of Biotechnology	ID/SER.O/13	45.00
Software Industry	ID/SER.O/14	25.00

Forthcoming titles include:

Timber Contruction for Developing Countries Introduction to wood and timber engineering	ID/SER.O/6	10.00
Timber Construction for Developing Countries Structural timber and related products	ID/SER.O/7	20.00
Timber Construction for Developing Countries Durability and fire resistance	ID/SER.O/8	11.00
Timber Construction for Developing Countries Strength characteristics and design	ID/SER.O/9	16.00
Timber Construction for Developing Countries Applications and examples	ID/SER.O/10	10.00
Design and Manufacture of Bamboo and Rattan Furniture	ID/SER.O/12	25.00

Please add $US 2.50 per copy to cover postage and packing. Allow 4-6 weeks for delivery.

ORDER FORM

Please complete this form and return it to:

UNIDO Documents Unit (F-355)
Vienna International Centre
P.O. Box 300, A-1400 Vienna, Austria

Send me _____ copy/copies of _____

_____ (ID/SER.O/____) at $US_____/copy plus postage.

PAYMENT

☐ I enclose a cheque, money order or UNESCO coupon (obtainable from UNESCO offices worldwide) made payable to "UNIDO".

☐ I have made payment through the following UNIDO bank account: CA-BV, No. 29-05115 (ref. RB-7310000), Schottengasse 6, A-1010 Vienna, Austria.

Name _____

Address _____

Telephone _____ Telex _____ Cable _____ Fax _____

Note: Publications in this series may also be obtained from:

Sales Section
United Nations
Room DC2-0853
New York, N.Y. 10017, U.S.A.
Tel.: (212) 963-8302

Sales Unit
United Nations
Palais des Nations
CH-1211 Geneva 10, Switzerland
Tel.: (22) 34-60-11, ext. Bookshop