Interdepartmental Task Force on Transborder Data Flows : background papers

WORKING COPY ONLY

1. [Background papers]

REPORT ON SOFTWARE RELATED ISSUES

Interdepartmental Task Force

on

Transborder Data Flow

Economics Working Group

M. Harrop
Supply and Services Canada

December 1981

1. OBJECTIVE

As a result of technological developments, there is a strong possibility that fundamental changes will occur in the methods of software production, distribution and use. In order to attempt to put this into perspective and also into the context of TBDF activities, the Economic Working Group of the Task Force on TBDF created a subgroup to address these specific software issues.

The objective of this report is to provide:

(i) a profile of software activities in Canada;

(ii) an examination of the major trends and causative factors (technology, economic environment etc.); and

(iii) an analysis of the potential for the development of Canadian software activities.

Recommendations for action to address the problems identified in the body of this report will be issued separately.

2. SOFTWARE: ITS NATURE AND AVAILABILITY

2.1 Introduction

In the course of the study which led to this report, it became apparent that, while people engaged in computer software activities often assume that the fundamental nature of those activities is widely understood, this is frequently not the case. Even people employed in activities dependent upon software sometimes have only a vague idea of what is involved in its production.

It was, therefore, felt appropriate to begin this report with a very basic introduction to the nature of software - what it is and where it comes from. This, it is hoped, will help to promote a wider understanding of these activities and will also serve as a background for the sections which follow.

2.2 What is software?

A computer program is a series of coded instructions which define, ideally in a clear, logical, and unambiguous manner, the specific functions to be performed by a computer.

In order for a program to be executed by the computer, the instructions must be in the machine language of the computer's processing unit, i.e. the instruction representation must correspond to a form (normally binary) which can be directly converted by the processor to a series of logic steps. Although it is possible to write programs directly in machine language, the process of coding instructions as a series of ones and zeros is tedious, error prone, and makes trouble-shooting difficult. As computers have evolved, numerous programming languages have been developed which allow the user to communicate instructions to the computer in languages based on alpha-numeric forms rather than binary. An early step in this evolution was the development of assembler languages which consisted of mnemonic operation codes for which there was a one-to-one correspondence with the machine instructions. (As an example the programmer could use the mnemonic ADD rather than the binary form of 011010 to direct the processor to perform an addition.) Programs known as 'assemblers' were developed to convert these assembler language forms into the basic machine instructions which could then be submitted to the computer for execution.

As computers developed further, greater emphasis was placed on ease of communication with the computer and on improving programmer productivity. Greater functionality was added to assembler languages, e.g. by allowing a single mnemonic to be assembled into a series of machine instructions, and more complex (usually special purpose) languages were developed which allowed the programmers to write programs in a form more closely resembling English than the assembler languages. These more complex languages, of which COBOL (Common Business Oriented Language) and FORTRAN (Formula Translation Language) are common examples, became known as high level languages. The programs which are necessary to translate the high level language programs to machine instructions became known as compilers.[1]

Problem Oriented Languages (POLs) represent a further stage in the development of programming languages. POLs allow the user (who is usually a research professional - economist, engineer, statistician, etc.) to define complex algorithms directly using an English language-type syntax. POLs themselves are usually written in a high level language such as FORTRAN, often with supporting assembler routines.

The progression from basic machine language through assemblers and high level languages to Problem Oriented Languages is a logical one. It reflects the increasing emphasis placed on improving programmer productivity and ease of man-machine communication.

Until the mid-sixties, the division between the programming effort associated with computers and the actual computing machinery was fairly rigid: the computer was equipped with a fairly limited set of instructions and any additional functionality was developed externally by programming. Programs were stored on media such as punched cards, paper tape, magnetic tape or disc, and loaded into the computer's memory for execution.

During the mid to late sixties, microcode was introduced. Microcode is a series of machine instructions stored in the logic circuitry of the processor and forming a permanent program within the computer. It cannot normally be changed without replacing or modifying part of the electronic circuitry. In this sense, microcode acts to extend the functionality of the hardware. Microcode made it possible for users to have computer manufacturers include special function instructions (which were sometimes unique) or mini-programs in the instruction set of a particular machine. A further

development was the capability which enabled users to write and incorporate their own microcode as an extension of the basic instruction set of the machine. In some instances where the computer circuitry is dedicated to a single application, e.g. in the micro electronic chips used in calculators or electronic games, the microcode may be the only program which is ever executed.

Programs written and stored in microcode are usually referred to as 'Firmware'. Programs which are normally stored on external storage media (magnetic tape, disks, punched cards, etc.) and which are loaded from the external media into the computer for execution, are usually referred to as 'Software'.

In general, firmware programs are usually small and very specific. Software programs tend to be more general in nature and may, in some instances, be very large. Because of the fixed nature of firmware, errors must be detected and eliminated before the program is incorporated into the hardware circuitry, whereas software is not faced with this constraint. However, the processes of producing software and firmware are similar. These may include designing, coding, testing and documenting of the program. Whether a program is incorporated into microcode or is retained on a deck of cards, the programmer is drawn from the same pool of programming expertise.

One further point to note is that, since a program must be error free before being incorporated into microcode as firmware, it must go through a development and test stage during which time it is effectively software, i.e. a program which can be easily changed. Thus, firmware begins life as software and references to software throughout this report may, in general, be considered to apply to nascent firmware.

Note

1. For languages such as FORTRAN and COBOL, in which the program source code is converted to machine code (i.e. compiled) and then held in machine code for repeated execution, the program which does the translation to machine language is called a compiler. With some other high level languages, of which APL is a common example, the programs are not compiled. Instead, the program is held in source form until it is to be executed at which time it is interpreted line by line. In these cases, the translation program is called an interpreter.

2.3 Subdivisions of Software

Software is usually subdivided into system software and application software. System software comprises those programs which enable the basic computer to be used more easily. Included in this category would be operating systems, translation programs (assemblers, compilers, etc.), input/output control programs, testing and debugging aids, programmer utility programs and machine resource accounting routines. Much system software requires that the programmer have an intimate knowledge of the basic machine architecture and operation, together with access to privileged instructions.[1] In addition, certain portions of system code must often be executed with optimum speed and efficiency to avoid adverse timing effects. For these reasons, most system software has traditionally been written in assembler language.

Application software is generally considered to comprise those programs, or packages, which perform specific user-oriented data processing tasks. Examples of functions which may be handled by specific application programs or packages are payroll, inventory control, statistical analysis, linear programming, and reservation systems. Although some application programs are written in assembler language, the bulk of application programs are written in one of the high level languages. As the different high level languages have data structures and operations oriented towards the intended use of the language, e.g. COBOL is structured for business use, FORTRAN or PL/1 more for scientific use, the particular choice of language depends to a large extent on the type of application.

Between system software and application software, there is a further type of software package which, whilst operating at a higher level than the basic system software mentioned above, can perhaps be considered an extension of system software in that it is likely to provide facilities which are used by application programs. Included in this category would be Data Base Management Systems and Teleprocessing Monitors.

The above classifications of software relate to general purpose data processing. A further category of software not included in the above is the software written for a special purpose (often single purpose) computer or processor. This is usually known as embedded software, examples of which might be programs for the

control of artillery computers, spacecraft, electronic games or electronic telephone switches. With these systems, neither the computer nor the software is general purpose, the total package (hardware and software) often being produced as a turnkey system. The software is usually unique and written in the language best suited to the application and processor.

(Turnkey systems are total packages consisting of hardware and software, usually designed to perform a specific function. They are usually supplied by a software house or by a hardware manufacturer, with the supplier taking responsibility for the performance of the complete system.)

System and application software, being produced for a general data processing market, tend to be designed with those features likely to be of most use to the largest number of potential users. This can cause certain problems. Firstly, the user of a general purpose program will be offered a number of features for which he may have no use but for which he will have to pay both as part of the purchase/rental/maintenance costs and in running costs, e.g. in extra memory and longer execution times. (This situation may be acceptable except when memory is limited and/or execution time critical.) Secondly, the general purpose program offered may not have all the features needed. Thirdly, a particular application may be so individual that no packages are available. Where these problems exist, they are usually addressed by developing a new program (or package of programs) or by modifying the available general purpose programs. In either case, the result is what is known as custom or customized software. One problem with modifying packaged software is that the supplier of the package ceases to take responsibility for any problems which develop in the modified package and the installation thus effectively looses vendor support.

The bulk of the programming effort in user installations, as opposed to hardware or software supplier installations, is devoted to developing and maintaining custom software.

NOTES

1. Privileged instructions are instructions designed to protect the integrity of the system by controlling access to potentially disruptive operations. Access to such

instructions is normally strictly controlled by the operating system. Examples of privileged instructions are those instructions governing input/output, interrupts, and memory access and allocation.

2.4 Development and Programming Aspects

The choice of a language for the development of a program depends on many factors. It is necessary to consider, first of all, what languages are available on the particular machine (or timesharing service) on which the program is to be developed. Almost all machines offer an assembler language, and most offer a selection of higher level languages. Choice of a language will also depend on the nature of the application, the data structures required, portability requirements, the availability of suitably qualified programming staff and the specific applicability of available languages to the particular application.

Certain specialized requirements make the use of assembler language almost mandatory. These include the case where memory is extremely limited; where direct access to privileged and systems-type instructions is required; where complex interrupt handling code is needed; or where no higher level languages are available (as is the case with some mini-computers and microprocessors). However, the use of assembler language can have significant personnel and productivity implications, both during development and during the subsequent maintenance phase. Assembler programs traditionally involve significantly greater effort for development and testing than do higher level language programs to accomplish the same task. Personnel require a much greater amount of training to become proficient assembler programmers than to become proficient higher level language programmers. (Some of the high level languages are designed to be used by people with minimal computer knowledge.) Productivity of programmers working in high level languages is, overall, higher than the productivity of programmers working in assembler language.

Another factor to be considered is portability. Programs written in assembler language will usually be machine (or at least architecture) dependent. Higher level languages usually have some degree of standardization, although individual manufacturers may, in their implementation of a particular language, go beyond the recommendations of the standard for the language and thus introduce some machine or

installation dependent features. In spite of this, programs written in higher level languages tend to be somewhat more portable than assembler language programs.

Commercially available productivity aids tend to fall into two categories - procedural type languages and development methodologies. Procedural type languages may be either subsets of an existing high level language or they may be almost a special purpose high level language themselves. In either case, the objective is to allow a user to specify a problem to the computer as simply as possible by means of a series of procedural statements. Development methodologies, such as structured programming and proprietary design methods, concentrate on making the design, coding, testing and documentation (and, of course, subsequent maintenance) of programs a more orderly process. On the system side, productivity improvements are being made by improving user accessibility and availability (e.g. by time sharing access) and by general improvements which make systems easier to use.

2.5 Personnel Factors

Without going into too much detail at this stage, it is perhaps appropriate to discuss briefly some of the personnel factors peculiar to software production and maintenance.

The problems of shortage of skilled software manpower have been well publicized as have some of the productivity problems associated with producing software.

The activities related to the production of software, e.g. feasibility studies, systems analysis, design, programming and documentation, are all intellect-intensive activities and, as such, are generally unsuited to production line techniques. Specific educational requirements for programmers are generally less important than aptitude for the work. In addition to aptitude, certain other attributes may be desirable but, with the exception of a relatively small number of jobs requiring specific technical skills, most programming positions can be filled (and have been filled historically) by people coming from a wide range of educational and industrial backgrounds.

Programmers have traditionally been a very mobile workforce. This is probably due mainly to staff shortages and the resulting opportunities, particularly for the more junior levels, to change jobs and increase salary substantially after only a short time in a position.

Management of software production has been a historically difficult area, programming often being viewed as a task performed by brilliant but undisciplined, unconventional technicians practicing a craft. This view is changing and software production is becoming easier to manage, but the impression of the programmer as an unrestrained individualist who produces arcane algorithms is still common and sometimes valid.

Geographically, the greatest shortages of software staff have tended to be in the most heavily developed and industralized areas - in Canada, in the Montreal/Toronto/Ottawa triangle. With the increasingly heavy emphasis on remote computer access and with data communications and terminal facilities being plentiful, there is no longer a real requirement for the programmer to be physically close to the computer (except for certain systems programming tasks.)

2.6 Sources of Software

The most common sources of software are: in-house development; computer manufacturers; software houses; user exchanges; and turnkey systems vendors.

In-house Development

In-house development, either by on-strength staff or by contracted programming, has long been the main source of application programs and of custom modifications to existing software. Constantly increasing costs of both development and maintenance have encouraged data processing management to look for packaged solutions where possible. However, custom software retains a high level of popularity and in-house development remains the largest single source of software supply. This may be due, at least in part, to the "Not Invented Here" attitudes which exist and which are discussed in Section 4.

Computer Manufacturers

In order to do any useful work on a computer, various systems software functions must be available. Traditionally, the initial source of this software has been the manufacturer of the computer. The U.S., being the world's largest supplier of computer hardware, dominates this sector of the software market. In addition to operating system software and utilities, the manufacturers have been a traditional source of assemblers, compilers and interpreters. In addition, many manufacturers now offer a range of application programs. Although, until the early 1970s, most manufacturers' software was included in the price of the hardware, more recent 'unbundling' policies have resulted in software and hardware being separately priced. The net result of this 'unbundling' policy is that some software is supplied at no charge, some software is fully charged, and other software appears to be partially charged. Where the software is not fully charged, it may be assumed that such costs as distribution and maintenance are still included with the hardware charges.

Deficiencies in manufacturer-supplied software have, in the past, created opportunities for independent suppliers to produce alternative packages. Unbundling of software has encouraged buyers to shop around, also creating opportunities for independent suppliers.

Software Houses

Software houses which maintain a pool of expertise, usually with a broad range of experience of different machines and programming languages, customarily produce software packages, supply supplementary staff for in-house program development, or develop turnkey systems. In addition, they may act as sales agents for proprietary software packages produced in the company or elsewhere. In some cases, software houses have expanded into the service bureau or custom hardware area.

The initial cost of developing a software package can be very high, many man-years being required for a complex package. Without outside funding, many software houses must rely on their consulting and turnkey activities to fund development projects. The U.S. has enjoyed traditional success in software house activities. The U.K. has also been particularly successful in this area, but with the emphasis being on

turnkey systems and consulting rather than packaged software. Canadian companies are also enjoying considerable success in the turnkey and consulting areas, particularly in the U.S.

User Groups

Most manufacturers of computers sponsor user groups for their customers. Software exchange schemes are a useful aspect of the user group activities. In addition, software exchanges are often arranged formally or informally between members of other common interest groups. Under such exchanges, it is usual for both the manufacturer of the hardware and the source of the software to disclaim all responsibility for the package supplied, i.e. the person acquiring the software accepts all risks associated with the package and also agrees to do his own maintenance, as necessary.

Other Sources

In addition to the above sources, software may also be supplied by third party software brokers or by computer stores. Software brokers are often used by software producers who do not have their own marketing force. Computer stores concentrate mainly on the hobby, small business and educational market. Programs for these markets are also widely available direct from the programmer through mail order.

2.7 Software Support

Maintenance for standard packaged software is normally provided by the vendor and included in the rental price. Where a package is sold outright, maintenance may be included in the price for a specific period of time, or may be obtained from the vendor as part of a separate agreement.

Software purchased or leased from a manufacturer is usually maintained by the manufacturer at no extra cost on condition that:

(a) that the software is listed as "currently supported" and;

(b) that the user has not made any modification to the software.

Manufacturer support is also usually available for no-charge software. However, experience tends to indicate that, when corrective action is needed, fully charged software receives a swifter response than does no-charge software.

Maintenance of in-house software is, of course, the responsibility of the installation.

Maintenance of turnkey and contracted software is dependent upon individual contractual arrangements.

3. THE ECONOMICS OF SOFTWARE

3.1 Introduction

The objective of this section is to provide a quantitative review of software production and use. Included in this section are estimates of overall software activity in Canada.

During the preparation of these estimates, it was necessary to make assumptions concerning software value, software longevity, and maintenance effort. There were found to be no uniform standards for estimating these items. As there are several valid ways of looking at each, a discussion of the issues involved in each of these topics is included in this section.

Early in the study, it was recognized that quantitative information on the software industry was incomplete and that it would therefore not be possible to attach precise numbers to products or activities. Instead, it was decided to develop order-of-magnitude estimates and projections. The basis used for these estimates is the Computer/Communications Secretariat growth model (Ref. 4). Before deciding to accept the Growth Model as a basis, estimates were developed outside the model and the results compared with those of the model. The results obtained were found to be of the same order of magnitude and the model, which had been used for previous estimates and projections, was judged to be still valid.

Because of the lack of consistent and comprehensive statistics on data processing in general and on software in particular, it was necessary to draw on data from several sources each of which uses different techniques and categories for data collection and analysis. In some cases where a single source presents data covering a number of years, the techniques and categories used may have been refined or changed over the years. It must be emphasized that extreme caution is necessary in the use and comparison of data obtained from diverse sources. It is stressed that the figures presented in this report are intended to give order-of-magnitude estimates only and should not be regarded as definitive.

Because of the great diversity, both of the software being produced and of the companies producing it, it was decided that it would not be appropriate, in the time

available, to attempt any form of survey, there being a high risk that any such survey would have been unrepresentative of the industry as a whole. Instead, informal contacts were established with members of the software community in order to establish an exchange of views during the period of the study and during the preparation of this report.

3.2 The Valuation of Software

Some of the confusion over software valuation is illustrated by the following examples:

- Software is regarded as an intangible asset by bankers, financial institutions and auditors when analyzing a company's assets or compiling a balance sheet;

- Revenue Canada/Taxation treats software as a tangible asset and requires that software development costs be capitalized. Depreciation of up to 100% is allowed;

- Revenue Canada/Customs & Excise considers that systems software should be charged duty at the same rate as the hardware if, as often happens, the software is included in the price of the hardware. Application software, or system software which is sold separately from the hardware, is not dutiable, although duty is charged on the media on which the software enters the country. Thus, for software carried on a magnetic tape, the importer will pay duty only on the tape. For a program carried into the country on punched cards or received via telecommunications, no duty will be imposed;

- The Federal Government requires that software be treated as tangible property when licencing and exporting high technology products to eastern block countries;

- Software is not regarded as patentable subject matter in Canada but may be granted a copyright. Some software has been patented in the U.S. In the

U.K. some software produced by a government agency is distributed with a crown copyright.

- The U.S. government contends that software is intangible and therefore not eligible for investment tax credits.

There is no doubt that the confusion over whether software is tangible or intangible is a problem, particularly for small companies wishing to obtain financing for software development. This section discusses some of the factors involved in software valuation. During the discussion, it will be useful to bear in mind the difference between cost and value.

Market Valuation

Unless a software package is to be sold, it is difficult to attach a market valuation to it. When a software package is sold, its value to the vendor and/or producer is related to the revenue generated.

In arriving at a price for a software package, it may be assumed that development costs, marketing costs, projected sales volume and projected maintenance costs (for those packages sold inclusive of maintenance agreement) would be the primary determining factors. However, indications are that a frequent overriding factor is simple market economics, i.e. "what the market will bear". This helps to account for some of the erratic price movements, particularly for software which is sold rather than leased.

In some cases, particularly where software has been "unbundled", software packages which were previously included in the hardware price are now distributed "free". In addition, some new, basic software is also distributed and maintained at no charge. It is, however, generally assumed that costs of the "free" software are included in the hardware - a view supported by the RC/CE position on system software valuation. It is estimated that approximately 25% of the cost of the hardware can be attributed to this "free" software. (It should be noted that, in the case of IBM, the amount of useful "free" software is being progressively reduced. The process of achieving this includes making only very basic packages available at no cost and

charging for any performance related enhancement. The charges for these enhancements and the benefits resulting from them are such as to make it no longer cost effective to run an enhancement-free system.)

A further factor which influences the value of software to a vendor is the "silent salesman" factor, i.e. software distributed by a vendor is often designed in such a way as to encourage the user to add more equipment in order to optimize the use of the package.

Clearly the revenue generated by the software package alone may not give an accurate indication of the value of the package, even to the producer.

Development Cost Valuation

For an installation contracting out a software development project or even developing software in-house, the initial cost can be assessed fairly easily. However, the cost of ongoing maintenance and enhancements to the package tends to get absorbed into the general EDP budget and the total ongoing cost of the inhouse package is frequently underestimated or unknown.

Replacement Cost Valuation

Another way of valuing software is to consider the potential cost of replacement. In most cases (owing to inflation) this will be significantly higher than the original cost of developing the software. In some cases, however, replacement cost may be lower than the original cost, e.g. if an off-the-shelf package is now available to do the job, but was not available originally. Conversely, if a user were faced with replacing a package product with custom software, the increase in cost could be extremely high, particularly if the product replaced was a no-charge item.

Value in Use

This is possibly the most difficult way to assess software value but that does not make its consideration invalid. The concept here is that a program is valued according to the function which it performs, i.e. according to its operational effectiveness or the extent of its usefulness. Efficient programs which are used extensively, which generate significant revenue, either directly or indirectly, which result in significant savings or which perform some critical control or processing function, are clearly more valuable than programs which perform some trivial or non-essential task. For such programs, the initial purchase or development cost may bear no relationship to the importance or value of the function performed. In such cases, an assessment of the cost and consequences of not having the program may provide an indication of its value to the installation.

Although the foregoing does not claim to be a complete examination of the ways of valuing software, it serves to illustrate some of the difficulties faced in trying to determine the value, either of an individual program or of a national software inventory. It also serves to acknowledge that the method chosen for estimating the value of software activities in Canada (Section 3.6) is not the only method available although, given the circumstances and complexity of the issue, it is one which is believed to be realistic.

3.3 Software Maintenance Estimation

It is evident that a large portion of software activity is devoted to software maintenance. As there is no general agreement as to exactly what constitutes maintenance, the following discussion has been prepared.

There appear to be two principal ways of estimating software maintenance effort - one based on development costs, the other based on annual operation costs. Each of these methods can be subdivided into at least two ways of estimating maintenance costs, e.g. the former may estimate annual maintenance costs as a percentage of the cost of developing the software package or as a percentage of total life cycle cost (which includes the initial cost of development); the latter may express maintenance costs as a percentage of the annual EDP budget or as a percentage of the annual personnel costs.

Some development cost/life cycle estimates suggest that 30% of the total life cycle cost is usually assigned to systems development and 70% to maintenance (ref. 1). There are, however, several problems with the development percentage method of estimating maintenance effort. Development costs are usually given in actual cost dollars rather than current dollar figures. In times of high inflation, the actual cost of maintaining a software package may be considerably higher (in real dollars) than the development cost. Further, since inflation is not constant and future inflation rates cannot be accurately predicted, unless the development costs are revalued to current dollars each year, it will not be possible to obtain a reliable and consistent development/maintenance cost ratio. A further problem is that the development cost may be unknown, e.g. with a vendor-supplied operating system which may be a no-cost item but which may require regular and extensive maintenance activities. Other problems with this method of maintenance estimation are:

1. the maintenance activity tends not to be level over the life of a piece of software, but rather to be concentrated in the first year or two and then diminish, i.e. software should become more reliable with age. (This will tend to reduce but not eliminate the effects of a varying rate of inflation.);

2. well written packages require much less maintenance than poorly written packages. In theory, the more effort put into the development, the less maintenance should be needed, i.e. there should be some relationship of inverse proportionality between development effort and maintenance effort. This also makes it difficult to arrive at figures which indicate an average maintenance cost related to development costs; and

3. although some maintenance functions, e.g. routine debugging, may relate directly to the size and complexity of the original development package, other maintenance functions, e.g. parametric type changes such as may be part of the system generation function, bear very little relationship to the magnitude or cost of the original development.

Before examining the topic of maintenance as a percentage of current operating costs, it is appropriate to consider what exactly is meant by maintenance. Reference 3 suggests that of total person hours spent by programmers and systems

analysts, just under 50% is currently spent on application system maintenance, just over 40% on new application development. Other rough estimates have indicated a 50/50 split between development and maintenance, however this is in conflict with figures obtained from actual installations, some of which indicate that a much smaller or larger percentage of programmer/analyst expenditure is on maintenance. This discrepancy appears to be the result of differing interpretations of the term "maintenance". The functions listed under maintenance in Reference 3 and the percentage of man-hours expended on each function, are as follows:

emergency fixes	12%
routine debugging	9%
changes to data & input files	17%
changes to hardware & system software	6%
enhancements for users	42%
improvement of documentation related to enhancements	6%
re-coding to improve efficiency	4%
other	3%

From this it may be seen that the largest single activity is system enhancement. This leads to the question "What constitutes an enhancement?" An enhancement which adds new features without replacing any of the existing functions could be regarded as a development activity. In many cases, however, an enhancement may constitute the replacement or improvement of existing code. It would appear, therefore, that at least some enhancement activities should be included under development rather than maintenance. This would take into account the increased value added to the existing software base as a result of enhancements.

In considering maintenance as a percentage of current expenditures, perhaps we should consider what happens when expenditures change. In the normal course of events, total maintenance expenditures would increase each year to take account of any expansion in the software inventory. Distortions would arise when the total expenditures increased or decreased disproportionately. If expenditures were reduced, emergency trouble-shooting, routine debugging and generation of new systems would likely continue but enhancements and overall development activity would probably be reduced. In such a case, the percentage of maintenance activity, excluding enhancements, would be higher than average. (In certain circumstances the

enhancement effort may, however, be increased in an effort to prolong the life of the software.) With a disproportionate increase in expenditures, the converse would happen and maintenance activity, excluding enhancements, could be proportionately less. Such distortions would tend to average out over two or three years.

Since it is, in general, easier to reduce (or defer) hardware expenditures than people costs, it may be more appropriate to express maintenance activity (or expenditure) as a percentage of total software activity (or expenditure) rather than as a percentage of total EDP budget. For purposes of the TBDF study, this method is used as the basis of measurement of maintenance activity. If maintenance is considered to include all enhancements then maintenance activity probably absorbs about 66% of software resources in 1981, development only about 34%. If enhancements are excluded from maintenance then, according to the table from Reference 3, these percentages are reversed; development takes 66% of the software budget, maintenance only 34%.

3.4 Software Longevity

The Estimates of Software Activity (Section 3.6) make reference to a software life of 5 years and 10 years. Little quantitative data has been found on the subject of software longevity. It can be assumed that a software package which required a large development effort to produce it, will normally have a reasonably long life. It does not follow, however, that a program or package which required few resources to produce it will have a short life.

One factor influencing the life of a program is the industry sector for which the program is developed - certain sectors, e.g. scientific and the oil industry, often having requirements for one-shot or short term programs. System software, by contrast, tends to have a long life in most cases.

Another very important factor influencing software longevity is machine architecture. Fifteen or twenty years ago, a change in hardware usually necessitated large scale replacement of the software. With the more stabilized machine architectures of the last ten years and the efforts of hardware manufacturers to provide upward compatible ranges of processor, the potential life of a program is greatly extended.

A third factor is the trend to the use of standardized high level languages which also potentially prolongs the life of application software.

A complicating factor in trying to determine the longevity of a program is that, where programs have been in existence for several years, the current version may bear little resemblance to the original because of progressive changes and enhancements over the years.

A 1977 survey into software life expectations (Ref. 2) indicated that users expected the life of packaged software to be between 5 and 10 years (depending on the package and excepting operating systems which were estimated to last only 2-1/2 years). The suppliers of software packages estimated a life of 6 to 8 years for the packages and 4 years for the operating systems.

Reference 3 reports that in response to a 1978 survey of DPMA members, the mean age of application systems reported was 4 years, 9 months with the most frequently represented age category being 1 to 3 years. However, among the 487 responses, a significant number of old systems were reported, with 20 systems reported to be over 12 years old.

While accurate and meaningful quantitative information on software longevity is scarce, it would appear that, with the stabilizing influences of standard languages and machine architectures, the average life of a software application system or package (which was intended to be reusable, i.e. not one-shot) is likely to increase.

3.5 Factors Related to the Growth of Software Activities

The rapid growth of software activities in the last ten to fifteen years is a direct result of increased worldwide demand for computerized processing covering a wide variety of applications. This trend began in the mid 1950's, accelerated in the mid 1960's, as medium and large scale machines grew in popularity, and after slackening in the early 1970's, accelerated again in the late 1970's as the development of so-called minicomputers and microcomputers brought lower costs and made computing economically available to much wider markets.

The growth of the Canadian computer population is indicated in Fig. 3.5.1. The growth in value of computers installed in Canada is indicated by Fig. 3.5.2. Accurate figures for the world computer population are not available, however, some indication of world growth in computing equipment sales can be gathered from Fig. 3.5.3 which shows worldwide revenues of U.S. companies. (U.S. companies are estimated to supply 70% to 80% of the world's computing equipment.)

Throughout this high growth period, computing power has become significantly cheaper, mainly due to improved semiconductor technology (Figs. 3.5.5 and 3.5.6). As an example, $20, which will buy a 65,536 (64K) bit memory (RAM) chip today, could buy the equivalent of only a 1,024 (1K) bit memory in 1973. In addition, the relative computer power of the installed base was doubling (and continues to do so) every two to four years. In 1981 the relative power of the installed U.S. computer base was approximately 32 times that of the 1966 U.S. computer base (Fig. 3.5.7) and 10 to 12 times that of the 1971 base. As Fig. 3.5.4 indicates, Canadian expenditures on hardware have kept pace with the U.S. growth as indicated in Fig. 3.5.3. It may be assumed, therefore, with a reasonable confidence that the Canadian installed computer base has also increased in power at least 10 times over the last ten years. During this period, however, it is evident that growth of computer personnel has not kept pace with the increase in power of the installed computer base (Fig. 3.5.8). Figure 3.5.9 compares the growth rates of relative power, hardware expenditures and software staff and illustrates the main reason for the current shortage of software staff. An indication of the U.S. personnel situation is given by an estimate from SHARE, the IBM user group which claims that in 1975 there were 175,000 computers (minis, mainframes and small business machines, but excluding desktop machines) in the U.S., and about 220,000 programmers - a ratio of 1.3 programmers per machine. By 1980, this ratio had declined to 0.5 programmers per machine.

Throughout the last fifteen years, regular predictions have appeared to the effect that software packages would, within a very short time, account for a very large portion of software expenditures. In spite of some quite forceful arguments as to why this should happen, it has not happened. Packaged software still represents a relatively small (though growing) part of total EDP expenditures (Figs. 3.5.10 and 3.5.11). The bulk of software expenditure is still concentrated on custom software development and maintenance.

Total EDP expenditures are growing but the capital cost of the hardware is taking a decreasing part of those expenditures.

This decrease in hardware capital cost is offset to a certain extent by increases in the cost of maintenance. (The cost of maintaining computing equipment which needs frequent or prolonged human attention, e.g. electromechanical devices, can be expected to rise at a faster rate than its capital cost will fall. For electronic equipment which can be maintained by exchanging printed circuit boards which are then returned to a factory or workshop for maintenance, maintenance costs may be expected to rise less rapidly.)

Figure 3.5.1 - Estimated Numbers of Computers Installed in Canada 1970-1980

Monthly Rental	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980
Under $1,000	1300	2000	3200	4900	7100	9900	13400	17800	23200	29600	37000
Over $1,000	3160	3760	4340	5100	6500	8100	9700	11700	13600	15200	16800

Source: DOC estimate based on CIPS Computer Census and Other Sources

Figure 3.5.2 - Estimated Annual Rental Value of Computers in Canada 1970-1980

($C Millions)

	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980
Total annual revenue from											
(a) Monthly rental under $1,000	13	20	30	40	55	75	100	120	150	180	220
(b) Monthly rental over $1,000	387	435	490	565	665	755	860	990	1150	1300	1430

Source: DOC estimate

Figure 3.5.3 - Estimated Computer Hardware Revenues

$U.S. Millions

	1971	1976	1981
U.S. (Domestic)	6000	14000	23000
U.S. (Worldwide including Domestic)	11000	25000	48000

Source: AFIPS & IDC

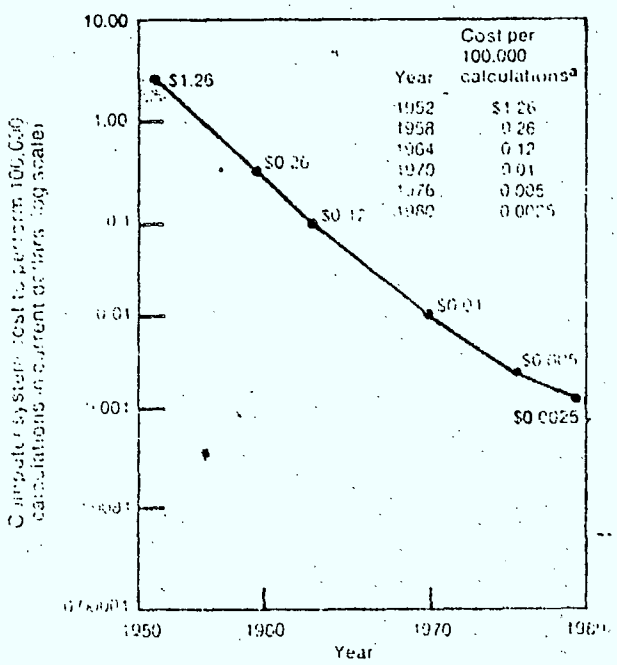
Figure 3.5.4 - Growth of Canadian Hardware Expenditures (Annual Rental Values)

$CDN Millions

	1971	1976	1981
Annual Rental Value (ARV)	455	960	1880

Source: DOC Estimate

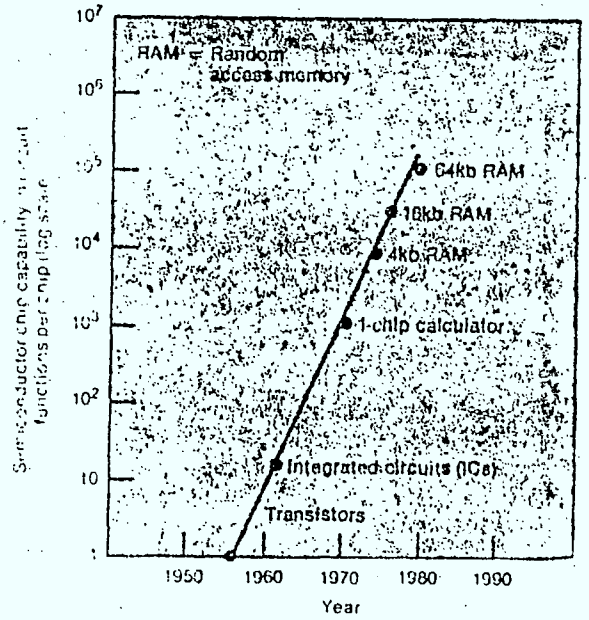
Figure 3.5.5 - Drop in Average Computer System Cost Per 100,000 Calculations 1952 to 1980



[a]Cost per 100,000 calculations is based on data for the following IBM computer systems (with year in parentheses): 701 (1952), 7090 (1958), 360/50 (1964), 370/155 (1970), 303 3 (1976), 4300 (1980).

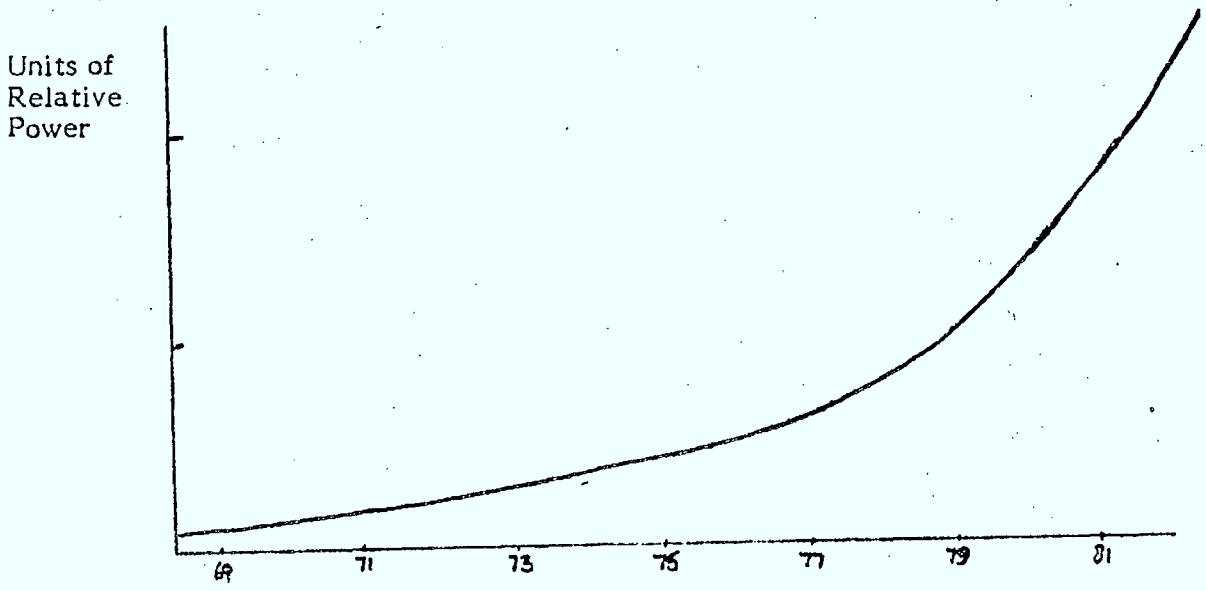
Source: Office of Technology Assessment - Ref. 19

Figure 3.5.6 - Increase in Capability of Semiconductor Chips 1956 to 1980



Source: IEEE - Ref. 20

Figure 3.5.7 - Relative Power of Installed U.S. Computer Base



Source: IDC - see refs. 12 & 13

Figure 3.5.8 - Growth of Computer Personnel in Canada

(Estimated Number Employed)

	1971	1975	1981
All Computer Staff	70,000	110,000	135,000
Systems Analysts/Programmers	20,000	33,000	41,000

Source: DOC Estimate

Figure 3.5.9 - Comparison of Growth Factors

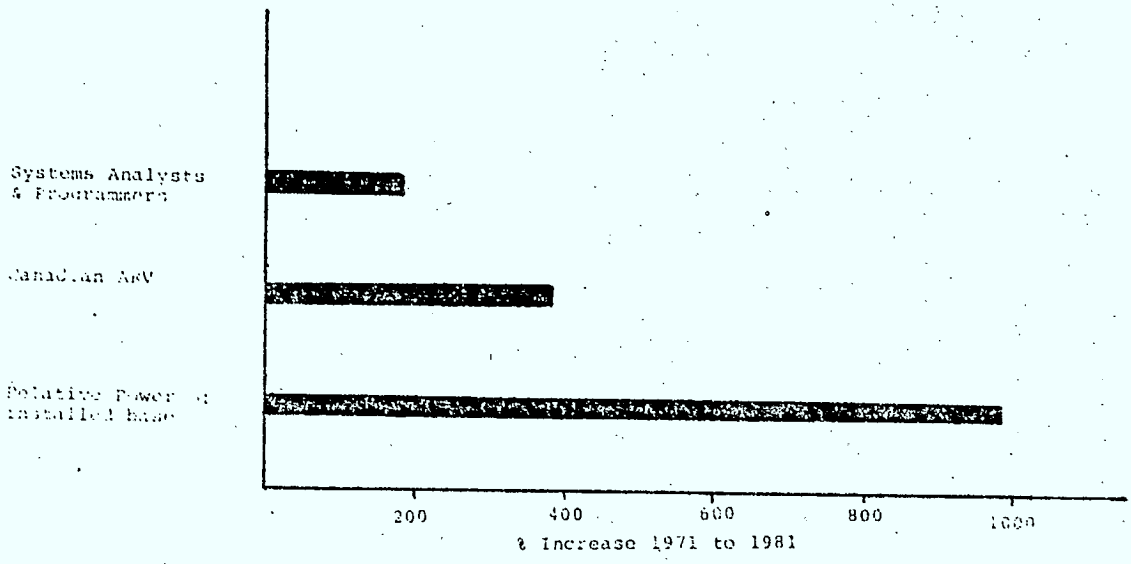


Figure 3.5.10 - Distribution of EDP Costs in Federal Government (by %)

Direct EDP Costs	1975-76	1977-78	1979-80	1980-81
Salaries & Consultants	43.4	44.7	42.3	42.7
Equipment Rental & Maintenance	21.9	19.8	21.4	21.4
Data Transmission	2.7	3.1	3.8	3.9
Service Bureaux	9.3	9.9	10.3	10.1
Software Acquisition	**0.2**	**0.5**	**0.6**	**0.6**
Production Supplies	2.8	2.4	2.4	2.4

Source: TBC-Ref. 7

Figure 3.5.11 - Computing Services Revenues in Canada (by %)

	1976	1977	1978	1979
Processing	65.7	63.3	64.1	64.1
Input Preparation	7.8	8.4	8.1	7.4
Systems Development	13.6	15.0	14.3	15.8
Software Packages	**5.5**	**6.0**	**6.9**	**6.9**
Other Services	7.5	6.6	6.5	6.0

Source: Evans Research Corp. - Ref. 14

3.6 Estimates of Software Activity

These estimates were made within the framework of the Computer Communications Secretariat "Growth" model (Ref. 4). This was done partly to make their context and limitations clear, and partly to facilitate comparisons with other estimates made on the same basis. It should be kept in mind that this is an **activity** model, not an **industry** model (Ref. 4, pp 8-11), i.e. it includes both in-house and market activity.

Software Development

At present, most computer software is written for the continuing use of a particular customer or firm (included in this category is the provision of customized packages for particular applications). The value of this software can most readily be estimated on the basis of the full cost of the resources used in its preparation (for in-house) or the revenues received from its sale (for contract work, etc.). Within the model format, Canadian users "own" as a resource the software that they have developed in-house or have paid to have developed for their use. They do not own software "rented" from manufacturers or software houses.

To estimate the value of software produced in-house by users of Canadian computing services, the following points were considered:

- CIPS Salary Surveys show that systems analysts and programmers (SAPS) account for about 35% of total EDP Personnel costs, and their supervision and management for a further 15%. Half of the Personnel costs established in the model can therefore be ascribed to software.

- Maintenance (including all enhancements) is considered to absorb the greater part of this staff cost today, but in the '60's most of this cost was used to develop new software. It is assumed that about 2/3 of this cost was available to develop new software of continuing value in 1965, that this share declined rapidly to 39% in 1975, more slowly to 34% in 1980, and stabilizes at 33% after 1985.

- Software development requires certain support costs. These are assumed to average 50% for machine costs, key punching and secretarial costs, and 25% for accommodation, supplies, education and travel.

- The resulting annual figure appears on the first line of section 1 of Fig. 3.6.1. Five year and ten year cumulations set the range for the value of software in use (first line, section 2). We have no solid information on how long a software system typically lasts before it must be re-developed, but 5-10 years seems a reasonable range. (See discussion in 3.4 above.)

Several assumptions also had to be made to set a value for software supplied by firms and individuals engaged in this activity:

- Statistics Canada data (Reference 6) for 1972-1979 were converted to an activity basis and adjusted for the effects of changes in coverage, data suppression, and shortfalls. These adjusted data suggest that software formed the following proportion of total computing service activity revenues:

	1972	1973	1974	1975	1976	1977	1978	1979
Annual	16.9	18.3	25.3	21.1	19.6	21.0	23.9	24.9
3-term MA		20.2	21.6	22.0	20.6	21.5	23.3	
5-term MA			20.2	21.1	22.2	22.1		

- This suggests little change in the software proportion of services during the early '70's but strong growth in the late '70's (paralleled by the rise of one software development company to a leading position among service industry firms, the biggest of which were previously all service bureaus). The "annual" data above, rounded (and smoothed in 1978) were used for 1975-79, and a declining rate of increase to 33% in 1985 and 38% in 1980 was assumed. These percentages were applied to the Canadian Computing Service Revenues of the model to obtain software revenues.

- Software firms perform maintenance as well as new development. For lack of good information, it is assumed that their proportion of new development is higher, but also declining from 80% in 1970 to 60% in 1990.

This percentage, applied to software revenues, gives line 2 of section 1 of Fig. 3.6.1; five and ten year cumulations give line 2 of section 2.

Software Maintenance

Maintenance, including all enhancements, accounts for most of the remaining in-house expenditure on SAPS (including overhead: 80% of the balance is assumed) and for most of the remaining expenditure on purchased software (a constant 10% of the total is allowed for "other" work). This "other" work, which is not included in the table, would include executive software groups (and supporting purchased expertise), one-shot experimental programming and other miscellaneous activities. The estimates for maintenance appear in Section 3 and 4 of Fig. 3.6.1.

Maintenance work must be considered in relation to the amount of software to be maintained. If a five-year software life is assumed, then maintenance costs per annum are 28%-32% of development costs, which seems high. If a ten-year life is assumed, then maintenance costs fall in the 17%-22% range, which seems more reasonable.

Note that one outcome of the assumptions made to this point is that maintenance costs rise as a proportion of inventory through the '70's, but decline in the '80's. This outcome is believed to be essential to the continued development of computing use though it must be admitted that the evidence for it is still non-existent.

Other Software

Certain software elements reported separately in both Treasury Board (Reference 7) and Statistics Canada surveys (Reference 6) were included in equipment costs when the Growth model was designed. These were the costs of acquisition of software packages. Prior to IBM's "unbundling" at the end of the '60's, almost all such software was usually supplied without charge by manufacturers, and the amount and quality of software supplied was a critical factor in computer choice. Even after unbundling it remained a critical factor, and industry and government personnel have indicated that manufacturers' software (whether or not charged for separately) was

usually included with equipment rentals in their accounts. What tended to be reported to Treasury Board or Statistics Canada was payments to other software suppliers (hence the small size of the amounts reported). Most such packages originate with foreign firms, though some are sold and serviced through Canadian agents.

The proportion of equipment costs that these packages form can easily be isolated and expressed as a percentage of the equipment totals for the Canadian Use and Computer Services modules of the model. The percentages are:

	74-5	75-6	76-7	77-8	78-9	79-80	80-81
TB	1.9	0.9	2.3	2.5	2.8	2.9	3.2
C.S. Industry	3.1	5.6	5.3	6.2			

These data show higher expenditures by the service industry than government users (about 5:2) and a rising trend for both. These assumptions underlie the figures in Section 5 of Fig. 3.6.1 and represent a gradually increasing proportion of the new software developed (section 1), rising from 3.4% in 1975 to 8.2% in 1990.

Another software element that should not be overlooked (though again not part of the "inventory") is the software component of the use of foreign services. This is largely software developed by parent companies for use of parent and subsidiary, but also includes the appreciable software cost of specialized computing services purchased by users from foreign firms not directly represented in Canada (real estate listings, credit checking, cablevision billing and database services are all examples as typical as remote computing). For lack of other information, this was taken to bear the same proportion to the total cost of use of foreign services as software bears to the total cost of using domestic services (excluding the specialized and one-shot jobs). This appears as section 6 of Fig. 3.6.1.

For Comparison

Two other lines attempt to give some perspective to these estimates. The first (Section 7 of Fig. 3.6.1) is the increase in annual rental value of computers installed in Canada. If this is compared to new software developed in the year (Section 1) the increase in computer rentals falls from 25% of new software development cost in 1975

to only 14% by 1990. The second (No. 8) is the undepreciated capital cost of all computers installed in Canada, which can be compared with the estimated development cost of all software in use (Section 2). If the longer 10-year average life of software is assumed, then software progresses from 89% of computer value in 1975 to 103% in 1990, i.e. it is of the same order of magnitude. If the shorter life cycle, it is approximately 60% of equipment inventory - in either case, an impressive figure which has been largely ignored to date.

Many observers have, for some years, been suggesting that packaged software may be about to overtake custom software as a revenue producer for some of the software houses. There are some indications that this may now be beginning to happen. For example, Systemhouse received revenue of $8.4M from package software and $20.9M from services and custom software in 1980. That company's 1981 figures are estimated to be $54M from packages and $36M from services. The software houses which are benefiting from this increase in business note that most of the sales are being made to non-technical end-users rather than to EDP professionals in existing data processing departments. However, the latest data available from Statistics Canada (1979) still shows software packages as accounting for less than 30% of software revenues, and little increase in this share since the mid 1970's.

Figure 3.6.1 - Summary of Software Estimates

	1975	1980	1985	1990
Software Development			($M)	
1. Written in year by				
Canadian service users	380	660	960	1,200
Cdn. Comp. Service Suppliers	60	170	330	460
Total	440	830	1,290	1,460
2. Total Value in use			($B)	
Canadian service users	1.5-2.2	2.7-4.1	4.2-6.9	5.5-9.7
Cdn. Comp. Service Suppliers	0.2-0.3	0.6-0.8	1.3-1.9	2.0-3.3
Total Inventory	1.7-2.5	3.3-4.9	5.5-8.8	7.5-13.0
Software Maintenance			($M)	
3. Total annual cost				
Canadian service users	470	1,020	1,560	1,950
Cdn. Comp. Service Suppliers	10	50	130	230
Total	480	1,070	1,690	2,180
4. Maintenance as % Inventory	28%-19%	32-22%	31%-19%	29%-17%
Other Software			($M)	
5. Cost of Rented packages				
Canadian service users	10	30	50	80
Cdn. Comp. Service Suppliers	5	10	30	40
Total	15	40	80	120
6. Software component of For. Serv. Used by Foreign Service Users	90	330	950	1,620
For Comparison			($M)	
7. Increase in year of ARV of Computers installed in Canada	110	170	225	200
8. Undepreciated capital cost of all Computers installed in Canada	2.8	5.5	($B) 9.2	12.6

3.7 Desktop Computers

Desktop computers have been included as a separate topic for several reasons: they represent a specialized area of the computing market, an area which is getting a large amount of publicity and which is experiencing a high growth rate; the future orientation of this market is not yet clear; there is the potential for much greater diversification of software production; and both the merchandizing methods and the customer base for these machines appear to be quite different from those previously experienced in the computer world.

Although it appears that most desktop computers are sold for use as small business machines (SBMs) they are not normally included in the SBM category of computer surveys which tend to include larger machines with more peripherals and a higher price range than most desktops. Reference 8 suggests that purchase prices for SBMs are usually in the region of U.S. $5,000 to U.S. $100,000 and while acknowledging that many desktop machines are now being purchased as SBMs, it indicates that, when equipped for use as an SBM, the desktop computer is likely to have a price in excess of U.S. $5,000, rather than being in the usual desktop entry-level price range of U.S. $500 to U.S. $1,000.

The definitions of a desktop computer are at least as varied as those of a minicomputer or an SBM. The usual price range of desktop computers is $1,000 to $10,000, however, some desktop computers are available for around $200, a price lower than some pocket computers and programmable calculators. Surveys on the number of desktop computers seldom make clear what machines are included. Figure 3.7.1 estimates worldwide growth of desktop machines based on the U.S. IDC figures. Fig. 3.7.2 depicts estimates of the Canadian desktop computer population. These estimates are based on the assumptions that the Canadian market for desktops developed slightly later than the U.S. market and still lags somewhat behind.

There is no sharp division between desktop computers and other computers but rather a continuum of processing power which ranges from the very largest general purpose computer to the simplest microprocessor. At the low end of this continuum we have the single purpose pre-programmed computer on a chip, which may be used for controlling the fuel supply in an automobile or controlling a microwave oven. Increasing in complexity we have electronic games and non (user) programmable

electronic calculators. The next step in the continuum is the programmable calculator and the pocket computer from which we progress through the wide range of available desktop computers to the more traditional data processing machines.

When desktop machines were first introduced to the market place, they were directed mainly towards the home and hobby market and marketed through retail stores and by mail order rather than by traditional computer sales techniques. Surveys have indicated, however, that the small business user has accounted for most of the sales since 1978 (Figure 3.7.3). The rapidly growing market in desktop machines is difficult to categorize and to quantify. The desktop machine which is sold for business use, however, is likely to be more fully equipped than one sold for home or hobby use. The average retail price of a business desktop machine in Canada is believed to be of the order of $4,000 (excluding software). Discussions with dealers and suppliers would suggest that between 30,000 and 35,000 such machines are likely to be sold in Canada in 1982.

In the context of programming resources, regardless of the amount of programming necessary, each of these machines draws the necessary programming talent from the same labour pool, no matter whether the programming involved is a one time unchanging program for a microcomputer in a toy or a game, a set of programs to be made available off-the-shelf for buyers of programmable calculators, or large scale system or application programs.

The software market for desktop computer products and services is different in several respects from the traditional software market. The large number of desktop machines and their varied use has resulted in countless software packages being developed, often by hobby computer owners, and sold either through the retail outlets or by mail order. Experiments are being conducted in Europe into the transmission of such programs using packet radio techniques. Many of the manufacturers of desktop machines obtain their software by employing freelance programmers to whom they pay royalties. In this way, they operate like book publishers. Because of the high production volumes (and sometimes because of the low overheads of the software producers) these packages sell very cheaply, sometimes for as low as $10 a copy.

IDC estimate that the value of the worldwide desktop computer software market will grow to $200 million (U.S.) this year from $35M in 1978. One example of a successful desktop software product is a business package called Visicalc. This package makes financial analysis and planning relatively simple for non-technical professionals. Introduced in 1979 by a company called Personal Software and priced at $150 to $200 per copy, the package sold over 25,000 copies in its first year and has, to date, sold over 100,000 copies. Total turnover of Personal Software was $4M in 1980 and is expected to grow to $12M-$15M in 1981. Another example is Digital Research of California which, in 6 years, has built up annual sales of $13M largely from the sale of CP/M, the most popular operating system for 8-bit micro computers.

One possible problem for the small business user is that of software maintenance. If a popular package from a relatively large supplier is being used, problems are likely to be detected early, reported to the supplier, and remedied fairly quickly. If, on the other hand, the software is a little used package, possibly from an unknown source, not only is there a potential problem getting the problems corrected, they may not even be detected for a considerable time, during which many potentially disastrous things could be happening to the data being processed. For a small business man, particularly, the proliferation of desktop software suppliers and the lack of any recourse in the event of problems, is a cause for concern.

At the present time, the volume of desktop software activities is small when compared to the total of all software activities. Although this is an area which is growing rapidly, it appears unlikely to overtake traditional software activities in volume in the near future. What is not yet clear with respect to desktop machines is whether they do, in fact, represent a whole new area of computing or whether, as seems more probable, what we are seeing is a continuation of the trend towards wider use of data processing. It does appear clear, however, that this sector of the computing market will more closely resemble conventional retail markets than the market for larger computer products.

Figure 3.7.1 - Worldwide Market for Desktop Computers

	Total Numbers in Use	Total Value Shipped US $ Million	Total Value In Use US $ Billion
1976	21,000	170	.3
1978	280,000	900	1.5
1980	1,300,000	2,300	5.5
Projections			
1981	2,100,000	4,500	9.8
1983	4,800,000	9,300	25.8
1985	9,300,000	17,000	55.6

NOTES: Based on U.S. manufacturers having 75% of World market.
These figures make no provision of the retirement of machines
Source: U.S. Figures, IDC (Reference 9)

Figure 3.7.2 - Estimated Canadian Desktop Computer Population

	Total Number In Use	Total Value In Use CDN $ Million
1976	400	6
1977	1,600	13
1978	8,700	49
1979	21,000	99
1980	44,000	170
Projection		
1981	62,000	230
1985	240,000	720
1990	720,000	2,500

Source: DOC Estimate

Figure 3.7.3 - Use of Desktop Computers

	1978	1980
Home and Hobby	22%	15%
Larger Companies	11%	9%
Small Business	56%	67%
Schools	11%	9%

Source: Datamation (Ref. 10)

3.8 Software Development Initiatives

Given the nature of software, organized development of new packages on a medium to large scale can be a high risk activity. In spite of the obvious growing need for packaged solutions, companies without large scale financial backing are often unwilling or unable to commit significant resources to speculative software package development. Outside financing is usually not easy to obtain because of the speculative nature of the venture and the reluctance of financial institutions to regard software as a tangible asset. This section reviews some of the assistance available to encourage software development in Canada. In addition, there is a brief examination of some interesting schemes being tried by other countries.

Canadian Initiatives

Federal Government assistance falls into two categories - aid, and tax write-offs. Under aid, the Department of Industry, Trade and Commerce has an Enterprise Development Program which, although not restricted to data processing, can be used to encourage some software development. Conditions attaching to this program include that the development project involve innovation and that it represent a significant burden on the company's financial resources. In addition, schemes are available to assist in the cost of marketing software products abroad. National Revenue/Taxation permits tax write-offs for software development, both for the manpower costs and the machine costs.

At least one Canadian company (Systemhouse) has raised money for software R&D by an equity issue. Another company, (Sydney Development Corporation), is currently raising $10 million for software package development by means of limited partnership units. This scheme is similar to oil drilling funds, MURBs or movie partnerships in that prospective investors agree to buy a minimum of 10 units ($10,000) in exchange for which they acquire a share in the profits of the development. Part of the partnership's development costs may be written off against personal income tax.

Foreign Initiatives

Most western governments have some sort of R&D development assistance available to encourage the development of new software, e.g. the U.K. has a software products scheme which is administered by the National Computer Centre and currently is paying about $2M per year in the form of non-repayable grants of up to 50% of development costs. Such schemes appear intended to encourage development of products within an existing industrial base. Other countries have taken a more aggressive approach to encourage the development of a software industry. The Irish Republic has announced an incentive program of direct cash grants to companies creating software jobs in Ireland, paying up to $10,000 for each job created. Singapore and Japan are cooperating to set up a centre of software expertise in Singapore which, it is intended, will compete for software contracts on a worldwide basis.

The developments in Japan and Singapore are particularly interesting. Reference 11 outlines the steps to be taken to develop a software industry in Singapore. These include generous tax incentives to companies developing software; the introduction of computer studies at all junior colleges and secondary schools; the setting up of computer training centres to produce programmers and to train non-computer professionals; and the importation of foreign computer companies and personnel to help develop local industry and also to help transmit software skills to the local population.

Japan is building a significant market for its software. In 1970 the Japanese Ministry of International Trade and Industry (MITI) created the Information Technology Promotion Agency to develop software. By 1976, MITI had created a pool of 17 software companies and 5 manufacturing companies in a joint project to develop and export software, half the cost being borne by the government.

4. ANALYSIS

4.1 Potential Implications of TBDF for Canadian Software Activities

At first sight, taking the definition of TBDF in its narrowest sense, i.e. the flow of computerized data over national boundaries, one could well ask the question "What has this to do with software?" A closer look at the issues, however, indicates that TBDF could have a very significant impact on our software activities.

The technology which facilitates computerized transborder data flow has developed to the point where large volumes of data can be transmitted rapidly to or from any part of the world. Terrestrial transmission media are no longer required. A satellite channel and an earth station can overcome geographical barriers and problems of distance. The data transmitted could be computer programs being supplied from a central origin, or could be data associated with programs developed remotely by linking the development computer with the software production centre. What the technology has done is to make it possible to develop and maintain software at the most suitable location, wherever that may be in the world. Factors to be considered in determining suitability of location include manpower availability, cost, timeliness, manageability and ongoing support. Any country which can develop a centre of software expertise providing timely delivery of competitively priced software, together with effective ongoing maintenance support, will be in a strong position to compete in world software markets.

Because of the generally recognized shortage of software personnel in Canada, it could be suggested that any attempt to create such a centre of expertise here is doomed to failure. It must be recognized, however, that the software personnel shortage is not unique to Canada but is a world-wide shortage. Indeed, those countries offering an environment less hospitable to programming staff than Canada are suffering far more from the effects of this manpower shortage and are, as a result, being forced to pay extremely high prices to attract temporary programming staff. In many respects, Canada is in an excellent position to develop as a centre of software expertise. The necessary technological and educational infrastructures are already in place. The problem is that this has not yet been recognized politically as an urgent requirement for future economic development.

As Section 3.6 indicates, Canada has a very large investment in existing software and demand for new software (packaged and custom) will continue to increase for the foreseeable future. Unless we are able to develop the capability to meet the demand for software and related services, it is inevitable that the demand will be met by imports. On the other hand, if we are able to develop our domestic capabilities to meet the home demand, we will have the potential to become suppliers to the world, since it is part of the nature of software that once a program is produced, it can be replicated at very little cost.

The technology which facilitates TBDF could make us net exporters or importers of software depending on our ability and political will to meet these challenges.

4.2 The Need for a Canadian Software Industry

The software industry, unlike the computer hardware industry, is not dominated by one or two suppliers, but comprises many companies of greatly varying size, background and degree of geographic distribution. Reference 15 lists over 400 companies offering software products and over 70 companies offering consulting services in Canada. Acknowledgement of the diverse origins of software is made by reference in this report to software activities rather than a software industry. From a macro-economic standpoint, however, it is necessary to consider the totality of software activities as an industry, an industry with two components - software production and software services. In this respect, the software industry is similar to many other industries where a product is produced, marketed and serviced.

It is considered essential that software be regarded as a product. It is the software which transforms a naked piece of electronic equipment into a useful computing device. Analogies are many, but a computer with no software may be compared with a newspaper with blank pages or a television station with no programs. The resultant economic value of the end product is largely dependent on a component which cannot be touched or measured in the way the media on which it is carried can be touched or measured.

In addition to acknowledging that software is a product, it is vital that the economic implications of the current investment, and the anticipated increase in the use of software, be recognized. Any extension in the number of computers and/or computer-based applications results in an increased demand for software products and services. Reference 16 indicates that the computer population will continue to grow at a significant rate during the next eight years.

There are strong reasons for having software developed close to the market. These include gaining a sufficient understanding of user needs in order to produce a useful and relevant product, and providing strong post-sales support (training, problem diagnosis and consultancy). Software packages which do not fully meet user requirements are often procured as a starting point in a software development. In such cases, the cost of the effort expended developing or enhancing the package may greatly exceed the purchase price or even the original development cost of the package.

As the major hardware manufacturers' pricing policies change to more closely reflect the cost of the hardware, a corresponding increase in the rental and purchase price of software offered by the manufacturers is inevitable. This will afford considerable opportunity to independent software suppliers to offer alternatives to the vendor-produced software packages. These opportunities will be in addition to those created by the growth in demand for software outlined in Section 3. Much has been heard in the past about the lack of a Canadian computer hardware industry. In fact, the amount being spent each year on software development and maintenance, greatly exceeds the annual rental value of computers installed in Canada (Fig. 3.6.1) and this gap is likely to widen as hardware manufacturers fully price their software.

User needs make a strong, home-based software industry desirable; the economics of the situation make it essential.

4.3 Factors Influencing the Development of a Software Industry

Personnel Factors

In addressing the question of shortage of qualified software staff, it must be recognized that increasing the number of computer science places available in the universities and community colleges is not going to resolve the present acute manpower shortage. Even if the facilities and teaching staff (who are also in short supply in this area) were available to double, or even quadruple, the intake of computer science students, there would still be a manpower shortage in addition to which any such increase would take at least six years to have any appreciable effect.

It is recognized that a certain number of computer science specialists are needed each year. However, the majority of programming jobs do not require this degree of specialization. Indeed, for many applications programming positions, someone having a degree in computer science would be over-qualified and probably over-specialized. As noted previously, for most programming positions, aptitude is a far more important qualification than a particular academic diploma. Recognition that one does not need a degree in computer science in order to become a programmer is essential as a first step towards resolving the manpower shortage. Steps need to be taken to identify those with programming aptitude and to attract them to the profession. Educational institutions (including educational television), professional bodies, government and industry, all have an important role to play in increasing computer literacy at all levels. It is felt that effective on-the-job training schemes and manpower retraining schemes could be developed and encouraged on a much larger scale than is presently the case. One further point concerning the programs currently offered is that employees have indicated a marked preference for graduates of university and community college cooperative programs. These programs have been particularly successful in producing practical (as opposed to theoretical) computer professionals.

With the currently available technology, the computer can be taken to the programmer rather than vice versa - software development can be done in areas far removed from the main computer centres provided that adequate access to the computer is available via communication links. In this way, those areas where the software staff are more plentiful could help alleviate the acute shortage in other areas

without the personnel leaving the part of the country in which they have chosen to live. The main constraint on this type of decentralization is the requirement to maintain adequate local support in the markets being served.

Another way in which technology can help in alleviating the personnel crisis is in helping to mobilize those unable or unwilling to regularly attend a central work place. Given the opportunity and electronic aides, many disabled people are able to do a competent programming job from their homes. In addition, many women who have left the work force to look after homes or children, would welcome the opportunity to be able to continue employment from a home base on a full- or part-time basis. Schemes to encourage such employment are successfully operating abroad and could be adopted in Canada.

Of crucial importance to the success of any schemes for augmenting and mobilizing the programming workforce, is effective management. Possibly the greatest part of the software manpower crisis is the shortage of project managers, many programmers having little inclination towards management activities.

One further item included here under personnel but which could have been considered under education or productivity, is that of productivity aids for non-technical people. By improving overall computer literacy, and by devising more aids to enable non-computer people to use the computer themselves to define solutions to their problems, we shall reduce the need for application programmers. The problem here, of course, is that most of the aids needed are dependent on software for their development and maintenance.

Productivity and Portability

It is generally agreed that improvements in programmer productivity are necessary. Few reports, however, clearly define what is meant by programmer productivity. As this report is intended to be a discussion of software issues rather than a treatise on productivity, it respects that tradition, however, some discussion of common productivity measures is believed to be useful here.

The most common productivity concept is that of labour productivity, usually defined as the total output of an industry or factory divided by the number of workers employed to produce that output. This concept is not widely used in connection with computing (partly for lack of a suitable measure of total output), although it is interesting to note that it would likely show a sharp year to year increase in the productivity of computing-related staff for most of the period that computers have been in use. The concept which is used here is much narrower, and relates not to the final output of the computing process but only to the direct output of the programmer himself, expressed as programs completed, as lines of code written or, most often, as debugged (i.e. error-free) statements of code produced.

Measured in this latter way, there has been little change in programmer productivity over time. On the average, a programmer will produce some 10 to 15 error-free statements per day. (Although these figures may appear low, they include the time required to design, test, debug and document the program, as well as actual coding time.) However, 10 statements written in a high level language can usually accomplish much more than 10 statements written in assembler, which supports the general view that high level languages have increased programmer productivity, even if there has been no increase in line of code written.

Attempts to assess programmer productivity based on lines of code are complicated by an effect similar to the Hawthorne effect, such that if a programmer is aware he is being assessed, he tends to write the same programs using more lines of code than if he were not being assessed. In addition, most studies are based on large projects using quite a number of programmers. Although the figures represent an average, individual performances can differ greatly. A study in 1967 (Ref. 17) reported that some programmers are able to produce code almost 10 times more efficiently than others and that there was no significant correlation with length of experience or aptitude test scores and performance.

There is no consistent means of assessing productivity. Each study on the subject must decide what factors are to be considered, e.g. how much overhead is to be included and whether only programmers are to be included, or other support staff also. Programs of different types may not require comparable resources per line of code. This makes comparison between studies difficult and even raises questions about the validity of some studies. With so many possible variants in the parameters used to

measure productivity, it is not surprising that estimates should show considerable differences.

Perhaps some of the comments which have been made about failure to improve programmer productivity are unfair to both the programmers who, for the most part, are working with the best tools available to them and whose numbers have not kept pace proportionately with the number and power of computers, and to the industry as a whole which has been trying to cope with quantum leaps in demand for its services. It is evident, however, that insufficient attention has been given to improving the output of systems analysts and programmers. Such improvement could greatly alleviate the present personnel shortage.

Improvements can and should be made at both the hardware and software level. The user interface must be improved to enable the non-computer professional to use the computer directly, without needing the services of a systems analyst and programmer to translate his problem into computer language. Greater availability and use of interactive facilities would reduce turnaround time for the compilation and testing of programs. The use of standardized languages and improved development and testing methodologies would shorten development time and improve maintainability of programs. Improvements in machine architectures and greater use of virtual-type operating systems would enable systems programmers to develop software in parallel with other computer activities instead of needing a dedicated machine which is probably available for only a few hours each week and then at unsocial times. Also additional work needs to be done on development of standardized interfaces which would facilitate program portability by reducing the degree of architectural dependence of programs. One further possible area for improvement concerns how programmers spend their time. It has been estimated (Reference 18) that the average programmer spends only 27% of his time on programming activities, the remainder being spent on clerical and support functions and attending meetings, etc. It may be possible to improve productivity by transferring some of these mundane clerical and support functions to lesser qualified staff.

Part of the productivity question could be addressed by the increased use of packaged software. Two problems which exist with packaged software are suitability and portability. On the question of suitability, management and users could show a greater flexibility and willingness to compromise by perhaps admitting slight changes

in a requirement if it enabled an existing package to be used. The question of portability is less easy to address. There is a definite need to concentrate on portability aspects not only of packaged software but also of operating systems and programming languages. Portability factors must be considered at the development stage of a project. Usually, however, there is insufficient incentive at this stage to develop a portable system and the increased long term convenience of portability is sacrificed because of manpower or time shortages.

It is often believed that by programming in a standardized language, such as COBOL, portability will be achieved. This is, however, not necessarily the case. As mentioned in Part 2, where standardized languages are offered, the vendor invariably also offers extensions which the programmer finds irresistable. In practice, it is often necessary to redesign programs which have been written in a supposedly standard language in order to move from one family of computers to another. In such cases, the original program is able to serve as little more than a design specification for the re-written version.

Urgent review is needed of the whole area of software productivity and portability together with specific initiatives for productivity improvement.

Attitudes and Myths

No analysis of this topic would be complete without a comment on some of the attitudes towards software and its production - attitudes of both people engaged in software activities and people who commission software.

Myth Number 1 - "Software is Cheap". This myth arose in part because of the bundling and partial bundling techniques of the mainframe computer vendors. When the cost of software was wholly or partially concealed in the cost of hardware, no one really considered what the true cost of software was. With independently produced and marketed software, the total costs of producing, maintaining and distributing the package is passed on to the consumer.

Myth Number 2 - "Software isn't any good if its not invented here". This myth has resulted in countless hours of unnecessary programming effort to duplicate,

or almost duplicate, software which was readily available. Software staff encourage the 'not invented here' syndrome because, like the program bug, it gives them job security and makes for more interesting work than simply installing, or perhaps tailoring, someone else's code. Management often accepts this attitude either because of fear of offending the programmers (and perhaps driving them to alternate employment) or because of inability to find any valid counter arguments. Acceptance of this attitude contributes to the shortage of programmers and to some of the low productivity claims.

Myth Number 3 - "If no one else has done it, neither should we" or "Wait until someone else does it". This is an unfortunate attitude possibly stemming from lack of confidence. It is an attitude all too familiar to Canadian product developers who, unable to get backing for a product in this country, are forced to take their ideas and expertise abroad. Paradoxically, this is an attitude which coexists with myth number 2. It is, however, an attitude which deters innovative software development in this country.

Myth Number 4a - "Programming is Difficult". Myth Number 4b - "A program must be complex to be good". These two myths are promoted by members of the "black art" school of programming who believe in maintaining programming as an elitist activity and who also believe that a program should be a testimony to the cleverness of the writer. Anyone who has tried to debug or modify a program written by one of these programmers will support the early dissipation of this myth.

Elimination of these attitudes or myths is dependent in part on the education process, i.e. increasing overall computer literacy, and in part on improved management control and understanding.

The Role of Government

In recent years, much of the focus of government aid has been directed towards making more competitive, those industries which make tangible objects. In industry today, "more competitive" often means "more automated" or "highly capitalized". Most segments of the computer manufacturing industry are highly capitalized. With the increasing trend to robotics, the need for production line workers

will inevitably decrease. On the other hand, software, which is by its very nature intellect-intensive, will continue to make heavy demands on the labour force. Indeed, without software, it would not be possible to render industry more competitive by the increasing use of automation. In addition, each software job created, results in at least one additional job in software support, e.g. data entry, secretarial work, marketing, management, etc.

Although it has not been possible to obtain confirming data, it seems probable that, in those industries engaged in the production and sale of data processing equipment in Canada, more people are employed in software and supporting activities than are actually employed in manufacturing the equipment, and yet strong bias is perceived towards aiding the manufacture of tangible goods rather than the production of items such as software. This is a bias which is in urgent need of adjustment.

It is generally accepted that the most successful, innovative and efficient software has traditionally been produced by small teams rather than large bureaucratic organizations. Any action which reduces the opportunity for small-team software development may well, therefore, prove counter-productive.

While direct grants do have a role to play in the development of some venture software, it must be recognized that, given the magnitude of software activities, the effect of direct grants is inevitably going to be small. Further, while not wishing to underestimate the role of direct assistance, there appears to be a marked preference on the part of industry, for trade rather than handouts. On the subject of direct aid, concern has been expressed that no grants or loans are available for the development of competitive software packages. (The current IT&C scheme requires that a project be innovative.)

The production of complex software may take many man years of effort and be a substantial drain on corporate resources. In addition to the production costs, support costs, e.g. marketing, documentation, post sales support and training, can be considerable. It is necessary to recognize the support and consultancy aspects of software as useful and productive activities. As the software environment becomes increasingly complex, these activities will assume vital importance.

Through the tax system, the government has the ability to offer significant inducements to software development. An example is personal tax writeoffs available to limited partners investing in R&D schemes. Corporate tax incentives for R&D are available but are, in some cases, quite restrictive, e.g. in an R&D project, the computer hardware qualifies for a write-off only if the machine is dedicated to the R&D project. A machine used half the time for R&D and perhaps the remaining time for other work, does not now qualify for any write-off.

The machine-based processing cost represents a significant part of any software development. The import duties and taxes placed on imported hardware result in machine costs being significantly higher in Canada than in the U.S. This has long been of concern to Canadian service bureaux and software houses who are finding more and more that economics favour their expanding by setting up data centres outside Canada.

The implementation of the government's EDP policy on software results in a large volume of business being done via the National Master Standing Offer (NMSO). The NMSO makes it easy to acquire software personnel for projects up to $50,000 but most of the projects are small and project control for the most part remains the responsibility of the department. The NMSO, while it provides a useful function and serves both government and supplier, does very little to promote software as an industry, as it is oriented towards the procurement of people rather than solutions.

In addition to positive steps which governments may take to promote software as an industry, it is at least equally important that measures which could harm the growth of such an industry be avoided. The success of at least one Canadian service bureau and software supplier is dependent on the free flow of the software from the Canadian head office to the rest of the world, and also on the ability of foreign sites to be interconnected with Canadian data centres for diagnostic service. In addition, this company has been able to employ programmers who live in the location of their choice (often abroad) but who contribute to the development of Canadian software packages by remotely accessing the Canadian data centres.

Lastly, if this country is to succeed in developing as a viable centre of software expertise, it is believed to be essential that dialogue be maintained between government and industry and between the diverse groups in industry engaged in software activities.

SUMMARY

If Canada is to develop as a centre of software expertise, there must be a concerted effort towards that goal by government, by educational institutions, by those engaged in software production and by users of software.

Personnel and productivity problems must be addressed realistically, constraints on the development of a software industry removed, and positive steps taken to encourage the production, development and ongoing support of software in Canada for both domestic and foreign consumption.

If present trends continue, Canada could be faced with an annual software import cost of $2560M by 1990. By developing a Canadian software industry which is able to meet the home demand, not only can that import cost be avoided but, with very little extra cost and effort, much software developed for domestic use can be exported worldwide.

References

1. Missing Dimensions in Systems Engineering - Smiley. Computer Data, March '79.

2. Computer Software and packaged Services Market - Frost & Sullivan Inc. 1977.

3. Software Maintenance - A User/Management Tug of War - Lientz and Swanson -Data Management, April '79.

   (Reference 3 summarizes a report on a 1978 DPMS survey)

4. The Growth of Computer/Communications in Canada - Computer/Communications Secretariat 1978.

5. Datamation - Source Fig. 3.8.3.

6. Computer Service Industry - Statistics Canada 63.222 1975 to 1979.

7. Review of EDP and Telecommunications in the Government of Canada, TBC 1980.

8. What is a Small Business Computer? Severino. Data Management, November 1980.

9. EDP Industry Report - International Data Corporation, June 3, 1981.

10. Datamation - Hobby and Game Markets Fade - Seidman - April 1979.

11. "Report of the Committee on National Computerization" - Singapore 1980.

12. IDC Industry Report - 26 June 1981.

13. IDC Advertising Feature in Fortune, 20 April 1981.

14. EDP Indepth Reports - Evans Research Corp., April 1981.

15. Computer Data, July/August 1981.

16. Estimates of Costs of Computer Use to 1990 - DOC, December 1981.

17. Exploratory Experimental Studies Comparing On-line and Off-line Programming Performance - Sackman, Communications of the ACM, Vol. II, No. 1, January 1968.

18. On the Management of Computer Programming - G.F. Weinwurm, Auerbach 1970.

19. Office of Technology Assessment and President's Reorganization Project, Federal Data Processing Reorganization Study, Basic Report of the Science and Technology Team, Washington, D.C., September 1981.

20. Institute of Electrical and Electronic Engineers, IEEE Spectrum, Vol. 17, June 1980 and "VLSI/LSI" IEEE Spectrum, Vol. 18, January 1981.

DATE DUE

MAY 26 1994			