# SOFTWARE TECHNOLOGICAL CHANGE: AN ECONOMIC APPROACH

Josep Mª VEGARA-CARRIO

Stanford University
Universitat Autònoma de Barcelona

Stanford, may 1983

CONTENS



o o o

---


I am indebted to Natan Rosenberg for his detailed comments and criticism ; I would also thank José Viñals for his cooperation.

A first version of this paper was presented at the Workshop on Technological Innovation Project , Stanford University , april 1983.

## 1. Introduction

At present,a very common expression appears in professional computer journals and meetings: "software crisis".It is very frequent to emphasize the raising software costs as as a percentage of gross national product,the almost stagnant software productivity and the increasing cost per line of code,the raising software weight as compared to hardware costs,the shortage of programmers,etc.In fact,there are no accurate and periodical statistics; however,independently of the exact figures,trends appear to be as indicated.[1]

Computers are one of symbols and myths of our age.Hardware plays an open,first class role; software,on the contrary,is the hiden face of computers,playing an apparent secondary role; its immaterial and abstract nature makes difficult to understand its functions,economic characteristics and specific technological change.Software is different from cars,ships,ovens,turbines,aircrafts,buildings,machines tools,plastics and other physical devices.Software -and specially software technological change- deserve atention to discover its specific nature and features.

---

1. Probably one of the first contributions to the controversy on size and costs was Boehm(1973); according to author's estimation total software costs in the U.S. were over $10 billion, or over 1% the GNP.The author presented an Air Force estimate on Hardware-software distribution costs: software went from 15 % of total system costs in 1970 to over 90% in 1985 ("this trend is probably characteristic of other organizations,also", pg.48); the graphic has became very popular and it is very frequently reproduced in software engineering books and papers.

This generalization of Air Force data to other fields is not accepted as valid by other specialists: see Fox (1982),pg.22,

(cont.)

o o o

Sections 2 and 3 explore software economic and technological characteristics, specially those of large complex software and packages; Sections 5 and 6 analyse historical facts related to programming languages and systems software development and Section 7 synthesizes the crucial aspects of software technological change from an economic point of view.

o o o

---

(cont.)

Cragon (1982) among others, on the basis that data are specific to one kind of computer use (command and control); this criticism is reasonable.

Boehm's (1981 b) estimation for 1980 software costs was approximately $40 billion, 2% of GNP (pgs.17-18); the graphic is again reproduced.

Boehm's first (quoted) paper was discussed by Larsen (1973) and his quantitative estimations have been recently questioned by Cragon (1982). Phister, jr. (1979) and Electronics (1983) contain other estimations.

Reading the annual Review of IEEE Spectrum (published in january), for exemple, it becomes evident the increasing concern with software problems and their critical consideration.

"The computer revolution is running into a bottleneck that is beginning to slow its momentum. The stumpling block is software" Business Week, sept.1, 1980.

"The software design crisis, the extraordinary manpower cost of programming new information systems, could be eased by 1990"... Graham (1982)

There are more optimistis assessments:

"The much-publicized software crisis is far from over, but the past year saw a number of important steps taken toward a resolution", Electronics (1981)

## 2. Software characterization

Modern digital computers are based on the stored program approach developed mainly by J.von Neumann:[2] the crucial concept is to store the set of instructions to be executed in the computer memory,as information,and not to use wired physical connections for this function.[3] This is the basis of the characteristic functional flexibility of the general purpose digital computer: it is sufficient to change the program stored in the memory to modify computer operations.[4]

Stored program concept implies a neat differentiation between physical equipment (hardware) and stored instructions (software). Software is sometimes considered as equivalent to programs but I will take a lightly different approach: I shall define software as a program or a set of interrelated programs and the documentation necessary to communicate and use it..The inclusion of documentation in the definition is not a luxury; the usefulness of software and the possibility to enhance it depend crucially on the existence of complete documentation; furthermore -as we will see later- distribution of incomplete documentation is one of the procedures used to limit the ability of users to master completely software packages,preventing them of becoming vendors.

There is an important distinction between applications software and systems software.Applications software produces specific results,directly useful to the user (a payroll,an econometric estimation,information retrieval functions,etc);systems software

---

2-von Neumann (1945)

3-von Neumann paternity is not without discussion: J.P.Eckert,who developed with J.W.Mauchlay the ENIAC,has recently claimed that:

> "contrary to the popular opinion,the ENIAC did store encoded instructions.The reason for this misguided popular opinion,he argued,was that the coded instructions were stored by means not normally used for memory after ENIAC",Aspray (1982).

4-See Metrpolis et al. (1980) for a detailed history of early computer developments.

manages and controls computer operations but -by itself- does not perform functions directly useful to the final user (operating systems,complilers,etc.)

o o o

Whatever its material suport,software is immaterial:it is abstract information to process information;one direct implication of this feature is that software does not wear,with obvious consequences on maintenance activities.

Software is costly to develop -specially large scale software- but,on the other hand,reproduction cost is very low so that software developers have difficulties to prevent its appropriation by other persons or institutions because it is difficult for them to enforce their property rights.However,most sorftware is not produced as a public good; this is so because -as we will see below- there are technological factors related to compatibility between software and different hardwares limiting frequently the possibilities of using non authorized software.In addition,these last years there have been developed new legal protection means.[5]

Software problems are very different at the two ends of the size scale: on one hand hardware development and costs reductions have facilitated and reduced the costs of creating small programs (using hihg level programming languages,interactive computing,etc.); on the other hand,large software is more and more complex,raising

---

[5] See in Mooers (1975) the possibilities of copyrighting before the Federal Computer Software Copyright Act of 1980; this Act, and the protection it gives,is described in Greguras (1981).The changing and complex history of patent protection is analyzed in Frank(1979) and in Blumenthal(1983);Mossinghoff and Lawson (1981) contains the semi-official attitude of the Patent and Trademark Office.

increasing reliability problems and making its development more difficult and expensive.Consequently I will focus the analysis on large scale software characteristics and problems.

o o o

Large scale software raises specific problems.Large scale is not only a matter of counting the number of program instructions; it is also a matter of complexity which is related to the density and pattern of systems connections and,specially to the number of conditional or alternative branches (logical complexity)[6,7] since this feature produces highly combinatorial patterns with thousands or millions of alternative paths.On the contrary,a large,one-way-sequential or paralel program is not a complex software although it can take a long time to develop it and may also require a big team.

Belady and Lehman's approach[8] include the dimensions and complexities of the creation process in the characterization of large software and they identify variety and communication as the crucial features:

> "the root cause of the characteristics we shall identify is related to the concept of variety.A program is large if its code is so varied,so all embracing that the execution sequence may adapt itself to the potential variety of its operational environment: the specific input,the requested output and the environmental conditions in the executing system and in the user environment during execution.A program is large if it reflects within itself a variety of human interests and activities.And if it does then it will essentially lie beyond the intellectual grasp of a single individual.It will require an organized group of people to design,implement,maintain and enhance it.And it is the communication between the variety of activities implemented in the program,the communication within the implementing organization,the communication between the implementors and their product and finally the communication between all these and the organizational environment that lead to the emergence of the largeness characteristic..."[9]

---

6-See Titli (1979)

7-Fox (1982)

8-Belady and Lehman (1979),pgs.107-8

9-Belady and Lehman (1979),pg.111

Certainly,to limit complexity to software content implies a radical reduction of global,total complexity of large software creation process; this global process must be the focus of an analysis of software technological change.

Another important characteristic of large scale software is the absence of a developed theory,the lack of a well structured body of scientific knowledge:

> "There does not,at present,exists a general system theory or design methodology for complex systems. There is in fact some doubt whether a complete theory can ever be totally developed or discovered ...Nor are there,at least in the realm of computer software,systematic and complete methodologies for system specification and design.Even with the most meticulous requirement analysis,design and implementation process, the product as first released to its users will not,in general, possess precisely those functional characteristics and properties expected or desired in the application and user environment."[10]

As a consequence of these features,large-complex software cannot be implemented at once,it must be changed,it has to evolve; we will analyze later this characteristic in relation with software evolution and learning by using.

Complexity created by conditional branches means generally that there is an extraordinary number of alternatives to test, so that very often large software cannot be completely tested, given the available time and ressources.

Finally,the abstract and immaterial character of software is certainly shared by large-complex software so that the absence of physical constraints in design operates as an element which makes design,paradoxically,more difficult given that it implies the existence of a large number of alternative approaches. In physical systems,on the contrary,the design of one subsystem has implications for some other subsystems: it implies constraints in terms of physical interfaces,strength,heat insulating, etc.

---

[10] -Denicoff (1981),pgs.384-5.

Fox comparison between hardware and software is suggestive:

> "Hardware is a physical thing and is more limited than in its permutation and combination.We find instances where software is like hardware and should be developed in the same way.But in the very large systems,the two are fundamentally different. Here the hardware is like a piano: software is music.Hardware is a distionary;software is a novel".[11,12]

Software development process

The process of software creation may include important elements of research and invention (on numerical computational algorithms, efficient sorting procedures,pattern recognition techniques,etc.) but this is not the most common case: very frequently the "element of novelty"[13] is absent in the process or it has a reduced importance (this is often the case in current business applicatins) so we will concentrate our attention on the development process.

---

[11] -Fox (1982),pg.9

[12] -There are several extreme exemples of large scale software:

.the SABRE system,the first airline reservation system; its development lasts from 1954 to 1964; see Parker (1965)

.IBM/360 operating system: "at the peak over 1000 (people) were working on it -programmers,writers,machine operators, clerks,secretaries,support groups and so on.From 1963 through 1966 probably 5000 man-years weant into its design,construction and documentation",Brooks,jr.(1975),pg.31.

.The American Airline System (a new reservation system) has been working since 1979,connecting 13,500 terminals on real time and and up to 10,6 million messages per day; see Fox (1982),pg.5.

See other exemples in Belady and Lehman(1979),Fox(1982) and Boehm(1981 (1981 b).

[13] -Freeman (1974 b),pg.318.For a taxonomy problems (and related measurement issues) of scientific and technical activities, see Freeman(1974 a)

With the exception of research-intensive software, software works achieving the required technical performance specifications; uncertainty is concentrated on the efficiency of the performance, the timing of its completion and the ressources necessary to complete it.

Software development[14] requires a set of systematic steps; this is certainly a normative statement and it is not always applied in practice. Development process[15] implies six phases:

a) user's needs and software requirements identification;
b) general design;
c) programming and, eventually:
d) assembly;
e) testing;
f) documentation.

The process if frequently analyzed in technical terms but, in fact, when software implies different groups -as it is almost always the case with large scale software- the process is a political one: it involves power, conflicts, negotiations, etc.[16] This aspect is specially crucial for the initial stage.

User's needs identification is a complex process specially when there are several (identified) users, so that it is necessary to induce a compromise among different perceptions and interests. When multiple users are not previously known -as it is the case of potential users of a software package- the problem has a hihg level of uncertainty. As in any development process, user's needs identification is the less structured step, the most difficult to organize in a systematic way

---

[14] -In conputer science literature the most common denomination is "software engineering".

[15] -See Fox (1982), Gillet and Pollack (1982) and Jensen and Tonies (1979) among an increasing literature.

[16] -See Keen and Gerson (1977)

General design determines the main features of the project[17] to be programmed,tested and documented.Large software design is highly combinatorial: as already remarked there is a large number of feasible design alternatives which,in addition,are not physically constrained as it happens in material systems.However, although small projects are hardware-independent large project are,on the contrary,hardware-dependent: central process unit and memory utilization introduce external-global-harware-derived constraints[18]. So internal structure and design are physically unconstrained but global design can be under external and global constraints derived from hardware limitations.This combination makes specific designs specially difficult.

Programming and assembly is the equivalent to the "production" phase in material product development,so that after testing and documentation software development process generates a final "product" (and not an experimental one as it is generally the case in the material products development).On the other hand, the immaterial and abstract character of software eliminates all the problems related with "materials" and material science (stress, corrosion,fatigue,etc.) so important in physical products development.

Testing is a crucial step; there can be specification errors (uncomplete requirements,distorted specifications,etc.),design errors (missing functions,wrongly designed interfaces,etc.),

---

17 -H.A.Simon emphasizes design as the process of creating the artificial or,literally: "The Science of Design: Creating the Artificial" is the title of one chapter in Simon(1969),pg.55 and also: "design,on the other hand,is concerned with how things ought to be,with devising artifacts to attain goals",pg.59

18 -"Software Development of Large Systems is often Hardware Specific. Large systems usually require that the use of the hardware be optimized for economy and performance considerations.Central processor unit loading and memory requirements must be carefuly planned.This forces the designer to incorporate into the design of the program the hardware parameters and features of the system.This is not the case for the small program developer",Fox(1982) pg.8

programming and assembly errors (initialization errors,errors in loop controls,neglegted alternatives,etc) and also documentation error so that,often,final testing is preceded by verification of every step of the development process;[19] this is so,specially since the cost of errors in an increasing function of the degree of advancement of the development process.

Documentation is essential to communicate the abstract content of software;it makes also possible its continuus evolution;furthermore,its selective distribution is also an element contributing to enforce property rights on software packages.

o o o

Software development is a labor intensive activity; in spite of the recent development of a wide variety of methods and tools,[20]

---

19-All development steps are important for reliability : testing can only detect failure presence.There is a wide spectrum of reliability definitions;Kapetz'one emphasizes dependence on inputs:

> "The reliability of software is the probability that a software system fulfils its assigned task in a given environment for a predefined number of input cases,assuming that the hardware and input are free of error",Kopetz (1976),pg.10

20-Since the early seventies there has been a lot of work done on engineering and software development tools.

Randell (1979) is an analysis of the state of the art in 1968;Wasserman (1978) conains a brief history of software engineering and a prospective analysis.Jensen and Tonies (1979) Ch.4,contains a detailed exposition of structured programming;on this topic see also Gillet and Pollack (1982),Ch.4.

Wasserman and Gutz (1982) and Lerner (1982) are focused on the future of programming and on automatic programming,respectively.

the process is a non repetitive,non standardized,wide spectrum problem solving process.[21]

Labor ressources are measured in man-months[22],unit which suggests a high degree of substituibility between time and human effort,but as F.P.Brooks emphasized some years ago,the sequential constraints which often characterize software development means that labor intensive character does not imply flexibility in execution :

> "Cost does indeed vary as a product of the numer of men and the numer of month .Progress does not.Hence the man-month as a unit for measuring the size of a job is a dangereous and deceptive myth.It implies that men and months are interchangeable.
>
> Men and months are interchangeable commodities only when a task can be partitioned among many workers with no commu-

---

21 -There is a lot of factors contributing to determine the ressources necessary to develop software;those identified by Boehm in his(intermediate) model designed to estimate man-months are:

- .Product atributes: a) Required software reliability; b) Data Base Size; c) Product Complexity.
- .Computer Attributes: a) Execution Time Constraints; b) Main Storage Constraints; c) Virtual Machine Volatility.
- .Personal Attributes: a) Analyst Capability; b) Applications Experience; c) Programmers Capability; d) Virtual Machine Experience; e) Programming Language Experience;
- .Project Attributes: a)Modern Programming Practices; b)Use of Software Tools; c) Required Development Schedule.

This may suggest the complexity of the interacting factors contributing to determine the necessary man-months. Boehm(1981)pg.115-6

Note: in Boehm's words "for a given software product,the underlying virtual machine is the complex of hardware and software (OS,DBMS,etc.) it calls upon to accomplish its tasks"

22 -The common practice is to measure labor productivity in software development in terms of source instructions per man-month; this definition is not without difficulties,specially related to the measurement of efficient output; this topic is analyzed, among others,by Fox (1982)Ch.6,Boehm (1981a) and Radice (1982)

nication among them. This is true of reaping wheat or picking cotton; it is not even aproximately true of systems programming.

When a task cannot be partitionned because sequential constraints, the application of more effort has no effect on the schedule. The bearing of a child takes nine months, no matter how many women are assigned. Many software tasks have this characteristic because of the sequential nature of debugging"[23,24]

As a result of this almost general absence of substituibility, management of large scale software development is subject to strong rigidities so that it is very important to accurately forecast the necessary human ressources; otherwise development process can be subject to important delays. As a matter of fact large scale software development history is full of long and dramatic delays.

o o o

Software life cycle includes development, utilization and also evolution. Evolution is not maintenance in the repair conception since software does not wear out; evolution is adaptation and enhancement plus, obviously, error correction. More specifically, includes the continuous process of error detection and correction, adaptation to new data or to modified formats, and enhancement to perform new or more efficient functions, as an answer to new user's requirements. Evolution can also include adaptation of existing software to new or modified hardware.[25,26]

As I have remarked before, the absence of a complete and well developed theory and methodology of software development is one of the elements contributing to large systems evolutionary characteristics

The feasibility of evolution is a specific charactersistic of software related to its immaterial character; it is a specificity of software engineering in front of other kinds of engineering .

---

[23] - Brooks, jr. (1975), pgs. 16-17

[24] - Baldwin (1982) contains a recent discussion of Brooks' analysis.

When a design is materialized into a device,a structure,a physical system (an engine,a bridge,an aircraft) any attempt to modify it,to adapt or enhance it,implies a physical modification and this operation is generally expensive; to "erase" parts of the old system, in order to attach the new elements is really an expensive task;on the contrary,to erase instructions in a program is a relatively easy and cheap operation.Only radical modular design -with physically separable modules- makes possible evolution of physical systems inside the same vintage.

---

[25] -See Van Horn (1980)

[26] -According to Zelkovitz (1978) maintenance costs are 67 percent of total development costs; this is an indication of their crucial importance along software life cycle.

According to Lientz and Swanson(1980) the distribution of maintenance costs in a survey including 487 data processing centers (mainly business related) is (in terms of total number of man-hours:

| | |
|---|---|
| Emergency program fix | 12.4 % |
| Routine debugging | 9.3 % |
| Accomodation of changes,to input data and files | 17.4 % |
| Accomodation of changes to hardware,system software | 6.2 % |
| Enhancement for users | 41.8 % |
| Improve of program documentation | 5.5 % |
| Recoding for efficiency in computation | 4.0 % |
| Other | 3.4 % |
| | 100.0 % |

Separation between hardware and software makes easier software evolution; software flexibility is also an advantage, even for embedded (resident) applications software (aircraft navigation system, chemical process control system, etc.) since it means the possibility of enhancement without hardware modification

o o o

One common approach to deal with software complexity is modularization or modular methodology: the decomposition of the whole system into modules, functionally (almost) selfcontained software subsystems; this approach means that modules must perform a specific function and that their connections or interfaces must be minimal, as far as possible, one input and one output.

The advantages of modular approach are well known: it allows programmers to code one module without considering other modules (with the exception of interfaces); on the other hand, a module can be used several times in a program, in different sequences or loops, with no need to code it again. Modular testing is also easier since it can be implemented on a stepwise basis, testing first modules and, secondly, interfaces and the whole system. Evolution it is also easier since error correction and function expansion or modification can be done in a modular basis so that modifications can be limited to the appropriate modules; in other words, it means the possibility to develop a "modular tecnical change", incremental and flexible, tipical of complex modularized systems.[27]

---

[27] - Some software specialists emphasize the role of module construction has to play as building block creation in software development:

"Work in the 1980's in software engineering must focus on the construction of standardized, verified software components"; Wasserman (1978), pg.38.

Other specialists, on the contrary, consider that priority must be given to modular approach as a tool for complexity reduction in software development:

"Reduction of complexity rather than potential use in a general library, becomes the primary motivation in designing and constructing such modules. It is certainly not necessary to abandon the later objective completely", Gillet and Pollack (1982), pg.40

## 3. Standard software packages

Since software development costs are fixed, one obvious procedure to reduce unit costs is to sell it to several users; this approach is the foundation of standard software packages; packages are created and distributed by computer manufacturers or by independent software producers.[28]

Selling packages requires to develop software compatible with different systems; this kind of compatibility requieres a more general design than the strictly necessary to use with a specific system and means also adaptation of flexible parameters (input/output parameters, functions actually performed, etc) to the specific characteristics and systems requirements. Adaptation is possible because the design is flexible: in other words, since flexibility and adaptability was one of the objectives of the design during the development stage.

The adaptation effort necessary in every case is very different, depends on the kind of software; a linear programming package, for exemple, normally will require a small adaptation effort, basically to hardware characteristics, since the formal aspects of the procedure is environment-free; a payroll package, on the other hand, may require a lot of adaptation effort since the specific administrative rules may vary widely among organizations.

The best commercial packages are those with a large market; therefore they are those related to basic and general functions as data management systems, payroll, financial planning, statistical and optimization packages, etc. even if in some cases the adaptation effort can be important.[29]

---

28 -See Weil (1982), Ch.2; also see in Data Decisions (1982) a servey of U.S. software systems packages and in Data Decisions (1983) a survey of applications software.

Systems software is generally developed by computer manufacturers since it requires a deep knowlege of hardware characteristics; this,however,is changing in the microcomputers field where there are indepeudendent software producers whose operating systems are used by microcomputer manufacturers.[29]

IBM new unbundling pricing policy -induced by antitrust pressures,and applied since 1969-was a crucial factor for the creation of a large open software market; charging separately hardware,education and software,large software segments were opened to independent firm competition.

Packages have important economic advantages since pricing policies anticipate that a number of copies will be sold so that development costs will be amortized over hundreds or thousands of users. In addition,software package utilization implies that users have access to human skills that,otherwise,they should hire; package licensing allows therefore a smooth evolution of ressources and skills necessary to firms or institutions.

Software packages also mean its almost immediate availability without any need to organize an in-house development team which will spend months or years to complete the project.This timing aspect may be often crucial.Another important feature derives from the fact that vendors take the responsability of software evolution.Users,therefore,dont only license a package: they have acces to evolving software.

On the other hand,however,there are some drawbacks associated with package licensing.Since they are not custom-designed,packages are neither completely adapted to every user's requirements nor they are able to optimize every computer ressources pattern. Furthermore,packages means dependence on vendor: users have access to evolving software but they dont have the right to modify

---

29 -The most significative case is the MS-DOS developed by Microsoft for the IBM Personal Computer.

the package.However,the economic advantages are very often substantial and standard software package production is a growing activity.[30]

There is another interesting feature related to software packages.Since it is difficult to enforce software property rights, producers try to strngthen legal procedures introducing technical difficulties to copy the original package.On one hand,they generally distribute machine language copies which are difficult to understand,specially whether,on the other hand,producers add meaningless instruction to introduce noise into the program.In addition,as a common practice,software firms only distribute the documentation necessary to use the package but not the detailed information needed to modify it.As already mentioned,producers take the responsability of evolution,including enhancement;this is an additional feature strengthening property rigths since users,generally,dont want only acces to software but to evolutionary software ; obviously,free riders dont get these services.This operates also as a mechanism that strongly limits the interest users may have to copy the package and to distribute it,independently of legal constraints.

o o o

---

[30] See next page.

30 - IDC (International Data Corporation) U.S. software market estimation for 1981 was:

| Type of sales | 1981 ($millions) |
|---|---|
| Hardware manufacturers | |
| Systems | 1,155 |
| Utility | 805 |
| Applications | 235 |
| Subtotal | 2,195 |
| Independents | |
| Systems | 140 |
| Utility | 465 |
| Applications | 665 |
| Subtotal | 1,270 |
| TOTAL | 3,465 |

Quoted by Frank (1983) pg.183

Most of the hardware manufacturers sales correspond to IBM sales.

On independent companies Frank writes:"

"(they)are tipically small -(80 percent with sales of less than $5 millions per annuum and with fewer than 100 employees) to medium sized ($5 million to $15 millions).Only a few companies report their software product sales at $50 millions or more as,for exemple,Management Science America Inc. and Informatics General Corporation. Generally the smaller organizations one or two major products, wheras the larger companies market an integrated offering of families of products",pgs.182-3.

o o o

## 4. Learning by using and software

Since software development is a non-repetitive activity, learning by doing is not a (significative) phenomenon although there may be an increase in generic programming experience.

The interesting point, therefore, is to analyze the dimensions of learning by using as it has been formulated by N. Rosenberg.[31] In his approach, learning by using is characteristic of durable capital goods, complex, with strong "materials" or environmental problems which make very difficult to anticipate their performance. Furthermore, the functional features and specially their useful life can only be known by using them. Using is also the condition to design and implement optimal operating and maintenance procedures. Aircraft industry is the most typical case of learning by using; in addition, Rosenberg conjectures that the phenomenon is also found in other industries, among them in software creation:

> "The creative use of learning by using as a deliberate business strategy may now be an important factor in sone high technology industries. Consider the computer industry which in recent years, has relied increasing by on complex software products to make its systems useful to a broader range of users. The development of effective software is highly dependent upon user expirience. The modification of software systems in response to such user experience is now an intrinsic part of the process of software engineering. This is so because most software products admit a wide variation of inputs and processing options. The full range of these option cannot possibly be tested prior to the release of software. Thus the optimal design of software is one where the designing firm is dependent upon a flow of information from its customers. Furthermore, many computer companies routinely provide extensive software support which involves the modification of software when bugs are discovered bu users-as they inevitably are-during operation of the software product. The effectiveness of support services is improving the product after its release appears to be very important to the competitive success of computer firms. Such service arrangements represents an institutionalization of procedures for exploiting the learning by using phenomenon in the computer industry".[32]

---

31 -Rosenberg, (1982), Ch.6

32 -Rosenberg, (1982), pg.139

As already remarked,large scale software creation has not yet well developed foundations,a solid theory upon which to build its development; as a result,software performance is uncertain and there must be an evolutionary process.Large scale software is, generally, a very complex immaterial system; its crucial element of complexity is logical: it comes from conditional branches,from alternative paths in programs;these combinatorial patterns can lead to a very high number of alternatives and,as a matter of fact,it is practically impossible to test all of them; testing, therefore,cannot be completed,[33] cannot be exhaustive,given the available time and ressources.This fact has obvious consequences for large software reliability.

Software packages are increasingly used; a lot of them are used by hundreds or even thousands of customers.Packages are the main software field of learning by doing.As we know,one of its features is that package producers take the responsability of evolution so that they are in charge of error correction,adaptation to new data or formats,enhancement to perform new or more efficient functions and adaptation to new hardware.The only way users have to influence evolution is communicating their problems and needs to package developers; users learn (problems) by using; contractual specifications induce them to transmit this information to vendors who have to try to solve the problems;learning by using then completes the cycle when package developer implements the solution.

A wide and variate experience -coming from hundreds and thousands of users-means that an important percentage of all conditional branches are actually tested and,therefore,a high probability to detect errors or buggs however,it is necessary to emphasize that error correction by using means to catch them at the most expensive stage,that is to say,once software is already designed, programmed and even tested once.[34]

---

33 -"Anyone who insists on error-free large software systems does not understand software",Fox (1982),pg.189.

34 -See Boehm (1981a),pgs.39-41 on the relationship error costs-development steps."These factors combine to make the error significantly 100 time more expensive to correct in the maintenance phase on large projects than in the requirements phase",pg.40

Obviously, a permanent contact with a big number of users provides to packages developers a very useful information about their needs related with data structures, functions and new hardware adaptation; this information will be crucial not only for short run adaptations but also to design new package releases and specially to select the range of options to be supplied.

These comments emphasize the importance of the size of actual or potential markets[35] as one crucial factor defining the volume of learning activity; obviously, there are important differences among countries.

o o o

[35] Succesful Software Products
Volume Achievers

| Product | Vendor | Number of installations | Cumulative Revenue ($millions) | Installation Price ($) |
|---|---|---|---|---|
| The Librarian | Applied Data Research | 4,500 | 20 | 8,000 |
| Westinghouse Disk Utility | Westinghose | 3,522 | 1 | 2,500 |
| Panvalet | Ansophic Systems | 3,300 | 10 | 6,000 |
| CA-Sort | Computer Ass, Inc. | 3,000 | 10 | 5,000 |
| AutoflowII | Applied Data Res. | 2,100 | 10 | 7,500 |
| SPSS | SPSS, Inc. | 2,075 | 2 | 2,500 |
| Syncsort | Whitlow Computer Systems Inc. | 2,000 | 10 | 6,000 |
| Total | Cincom Systems, Inc | 2,000 | 50 | 40,000 |
| Fast Dump Restore | Innovation Data Processing | 2,100 | .5 | 4,000 |
| Alltax | Management Science America Inc. | 1,600 | 2 | 2,000 |
| Job Accounting Report System | Johnson Systems | 1,350 | 5 | 7,000 |
| Mark IV | Informatics, Inc | 1,300 | 50 | 42,500 |
| Easytrieve | Pansophic Systems | 1,650 | 10 | 15,000 |
| Ramis | Mathematica, Inc | 1,000 | 10 | 34,000 |
| General Ledger | Software International Corp. | 1,000 | 10 | 35,000 |
| Autolab II | Capex Corp | 1,000 | 1 | 14,450 |

As recognized by ICP, Inc.; quoted by Frank (1979)

o o o

There is another kind of learning by using in computer industry: its root cause is communication among computer manufacturers and their specific users groups (when they are effectively organized); the process includes software but also other aspects (hardware, education, etc.). Historically, interaction has included also high level programming languages; thus, Share formed in the early 1960s a Committee to improve the initial (Fortran) version. In 1963, IBM and the same IBM users' organization formed a Committee to develop a language more advanced than Fortran; Guide, another IBM users organization was later represented in the Committee; the result of this collective effort was PL/1, issued in 1965. USE, the Univac group, was involved in the development of Cobol.[36]

o o o

To conclude this topic it is useful to emphasize that there are obvious differences between large scale software and other systems showing learning by doing. The main one derives from the abstract, immaterial character of software; this means that software does not need maintenance in the physical-repair sense so that an important element of learning by doing -related with maintenance practices- is (missing). This characteristic also implies that there is no wearing and so no problem of (physical) life estimation problems; in other words, software has no material science problems; software support, however can be subject to these kind of problems, specially in the case of embedded systems operating in extreme environamental conditions).

o o o

---

[36]-SHARE and GUIDE are IBM users' groups; USE is the Univac users' group; VIM the Control Data Corporation group, etc. Almost all the groups maintain program libraries for exchange among members. On SHARE early history see Armer (1980).

There is also a global group, JUG (Joint Users Group), with a representative in the Committee X3 of the American National Standard Institute (ANSI), responsable for language standardization

## 5. Programming languages technical change

As it is well known, the first stored program computers had no systems software and applications software was directly written in "machine language", that is to say, in the binary code directly understandable by the computer. The first step away from machine language -that requires a good knowledge of hardware- was Assembler in which the operations to be performed recieve a mnemonic expression and memory adresses may recieve symbolic names; computer does the translation and finds the adresses. Computers can be programmed -given its logical and computational capabilities- to convert the assembler languange into machine language by using the most simple variant of languange processor or compiler.

Assembler -still in use- was the first move in the direction of the so called hihg level programming languages, characterized by their user-orientation, that is to say, the fact that their utilization does not require haruware or machine language knowledge; the expressions of hihg level programming languages statements are written in a formalized language well adapted to the specific features of the problem to be solved.

The invention and development of high level programming languages and the related compilers has been one of the crucial software innovations, simplifiying programming and rising productivity.

Before Fortran there had been several attempts in Europe and in the US to develop high level programming languages focused on scientific computation; some of them -only a few- were implemented. In 1953 J.H.Laning jr. and N.Zierler at MIT developed a language for Whirlwind; this

language "appears to be the first system in the United States to permit the user to write expressions in a notation similar to normal mathematical format"[37].J.E.Sammet qualifies this language as "probably the most significant of all the early work"[38]

The first widely accepted high level language[39] was Fortran, designed as a mathematical oriented language[40].The project was launched by IBM; the team was formed by a majority of in-house specialist and was headed by J.Backus.The language was issued in 1957;the first specifications are dated november 1954 and the first reference manual was issued in 1956.Fortran compiler was completed in 1957. Univac issued in 1961 the first non-IBM version and by 1963 almost all manufactures had or were developing their own Fortran version; a study published in 1964 mentions the existence of 43 Fortran compilers.

In 1958 was released a new IBM version -Fortran II- including among other additional elements,Function and Subroutine.Other updated versions have been implemented;Fortran is still widely used in scientific computation.[41,42]

---

37 -Sammet (1981),pg.204

38 -Sammet (1969),pg.132

39 -Detailed information on high level programming languages history and characteristics can be found in Sammet(1969),(1981) and Wexelblat(1981)

40 -In 1958 a team of european and american specialists developed a mathematical algorithmic oriented language called IAL; this programming language evolved into Algol 58 and Algol 60 which have been inflential on high level language evolution but without reaching wide diffusion.

41 -Fortran was not accepted without resistance: "although Fortran is considered quite common place now,it was not readily or easily accepted at that time.Customers raised many objections, foremost among them was that the compiler probably could not turn out object code as good as their best programmers"; Sammet (1969,pg.144)

42 -It was probably the biggest software produced till then; it had some 25,000 lines of code.

The economic motivation of Fortran is emphasized by J. Backus, the head of the team:

"Another factor which influence the development of FORTRAN was the economics of programming in 1954.The cost of programmers associated with a computer center was usually at least as great as the cost of the computer itself...

In addition,from one quarter to one-half of the computers time was spent in debugging.Thus programming and debugging accounted for as much as three-quarters of the cost of operating a computer; and obviously,as computers got cheaper, this situation would get worse.

This economic factor was one of the prime motivations which led me to propose the FORTRAN project in a letter to my boss,Cuthbert Hurd,in late 1953...I believe that the economic need for a system like FORTRAN was one of the reasons why IBM and my succesive bosses,Hurd,Charles De Carlo and John Mc Pherson,provided for our constantly expanding needs over the next five years without ever asking us to project or justify those needs in a formal budget".[43]

o o o

Flow-Matic was the tirst high level language adapted to the specific needs of business data processing; it was created by a team headed by Grace Hopper,from Remmington Rand Univac,and released in 1958; the language had a limited diffusion.The succesful initiative of developing a business-administration-oriented language was taken by the US Department of Defense in 1959;since Fortran was already available,it was a clear indication that it was considered as a specialized language,not suitable for business applications.The development team (the so called Short Range Committee) was composed by representatives of six different manufacturers ,officials of two government agencies and the chairman (member of the National Bureau of Standards). The new language was issued on december 1959; it was published on april 1960.The first two compilers were implemented by Remington Rand and RCA in 1960.Cobol is also still a widely used language in its specific field.

---

[43] -Wexelblat (1981),pgs.26/27 .Sammet (1981),pg.206,formulates a similar comment.

Since the early 1960s a lot of different programming languages have been developed.[44] New languages have been designed with the objective of being rather general (PL/1 and,recently,ADA, are the exceptions);in fact,an important almost general feature of high level programming languages is specialization,their special adaptation to a specific application area; I have already mentioned Fortran and Cobol;APT was designed in the 1950s for machine tool numerical control;LISP was created in the early sixties for artificial intelligence applications; SIMSCRIPT is a special language for simulation applications;other languages are designed for equipment testing,circuit design,civil engineering,etc.[45]

The root of this characteristic -specialization- is that different subsets of natural language,once formalized,are well suitable to describe and symbolically manipulate different kinds of phenomena (numerical computation,geometrical path description, linguistic structures,etc.),so that the corresponding formal languages are different;to try to develop a general language is not feasible or,even if it were, it would require an extraordinary memory size and a gigantic software development effort; both costs are unnecessary with specialized languages,designed for limited,highly specific goals.

---

44 -Active high level languages (U.S.) in 1977:

| Application area | |
|---|---|
| Numerical scientific | 20 |
| Business data processing | 4 |
| String and list processing | 11 |
| Formula manipulation | 10 |
| Multipurpose | 34 |
| Specialized applications | 90 |
| total | 169 |

Sammet(1981),pg.210

45 -Some specialized languages have made possible important innovations as it is the case of numerical control machine tools; without APT -developed at the MIT- well suitable to describe tridimensional forms and paths and machine tool operations, numerical control machines would have not been feasible

High level programming language development shows an interesting pattern: languages have been developed by very different groups or institutions;innovators have been manufacturers (Fortran 1),users and manufacturers (PL/1),users (Cobol,Ada),research laboratories (APT,LISP) or even individuals (APL,developed by K.Iverson;Pascal created by N.Wirth).

o o o

The development of high level programming languages,to be operational,requires the avalability of special systems software - language translators or compilers- converting original statements into machine code.Compilers use part of the main memory; the compilation process requires central process unit time and generates a set of machine instructions greater than the required to execute the functionally equivalent program directly written in machine code.High level languages therefore imply a trade-off between programmer's time,on one hand, and computer time and memory,on the other hand.In spite of these drawbacks,high level languages have been spread and their utilization is very general,unless there are tight time and memory constraints.They are easier to use than machine code;they are also easier to test and debugg than programs written in machine code and,as a result,their use implics higher productivity.

The difusion of high level programming languages has been fostered by hardware price reductions; that is one of the multiple exemples of interaction between hardware evolution and software.

o o o

Natural language -although it has,obviously,a structure- is not standardized but context and redundancy are generally sufficient to allow its understanding.High level programming langua-

ges have been standardized[46] since they are selfcontained and very poor compared with natural language. To be translated by compilers their structure must have a reduced variety; in addition, standardization simplifies all kind of communications; learning and training is easier; compatibility (portability) is improved and documentation production is simplified.[47] Standardization, however, is not without difficulties; as J.E. Sammet has emphasized:

> "one aspect of standardization that in my opinion has caused the most difficulties is the fact that proponents of languages continually wish to extend its scope so that it becomes more suitable for a larger class of problems. The most notable exemple of this have been BASIC and FORTRAN. The former was originally (designed as) a very simple language, but so many things have been added to it in to ongoing standardization process that one would hardly recognize it as being a simple language. The same extension mania applies to FORTRAN, and COBOL has also had significant additions over the years. One can, of course, argue that all these changes (not just additions) are in the name of progress, and as new capabilities and needs arise, then they should be put into the base language. In my view, those languages will eventually approach in complexity and size those languages which were designed from the very beginning to be large, namely PL/1, ALGOL 68, and Ada. Then we will have to create new simple languages. Unfortunately, the smaller languages (e.g., BASIC) will become large in a very ad hoc fashion, and as a result just cannot be as effective as a language designed from its inception as an integrated large language".[48]

Standardization has not been, therefore, powerful enough to enforce a reasonable evolution of programming languages, specially to prevent their almost general unplanned complexification.

o o o

---

46 - In the US the Committee on programming languages standardization -X3- was formed in the sixties inside the American National Standards Institute (ANSI). For a detailed description of standardization process see Hill and Meek (1982), Ch.1.

47 - For a deep analysis of language standardization problems see Hill and Meek (1982)

48 - Sammet (1981), pgs. 216-217.

## 6. Systems software technical change : operating systems.[49]

The first stored program digital computers executed a program at a time;programs were only applications programs.All computer ressources (central process unit,memories,input/output devices) were blocked by the running program. When execution was over the whole computer was idle while the operator got the next program ready for execution.The problem was similar to organizing the schedule of a set of non-automatic machine tools.

These evident and costly drawbacks prompted the development of systems software,more specifically,of operating systems.[50,51] The first problem solved -in the early 1950s and by one elementary software system- was to perform program transition minimizing idle time.It was first developed by users: General Motors created the first system and in 1955 GM and North American Aviation worked together to solve the idle time problem in a more advanced computer; their success was widely recognized and diffused.

As already remarked the problem was first solved by users:

> "The manufacturers have been quite slow in understanding the requirement.The original problem atacked by operating systems -minimizing idle time- is not really a computing problem at all; rather,it is a problem of installation management.It is a users problem and was solved by users"[52]

Systems software was upgraded,afterwards,to perform new functions related to automatic documentation and accounting,to keep

---

[49] -In fact,we will consider operating systems.

[50] -Rosin(1969) contains one of the few existing US systems software history; see also Dolotta et al.(1976);Weizer (1981) is also useful for the recent period.

[51] -There is no common terminology: Rosin denominates this system an operating system and Steel (1964) uses the name words.Kurzban et al.(1975) classifyis it as a monitor; Rosin considers executives,supervisors and operating systems as havin the same meaning; Kurzban st al. consider the sequence as a succesion of more complex systems.

[52] -Steel,(1964),pg.27.

track of computer activity. A new approach to idle time problem implied to read a batch of programs from magnetic tapes.[53]

New systems software development during 1959-61[54] was supported by hardware advances; data channels were designed as small computers with limited function, essentially to control different input/output devices so that it was possible to overlap input/output operations and central process unit operations. Simultaneously, interrupts provided the possibility of reporting the end of input/output or exceptional operations occurrence; their occurrence prompted a predetermined action.

The increasing number of functions performed by operating systems increased the ammount of main memory necessary reducing the memory space available for user's applications programs; the reduction of memory prices solved an important problem for systems software development.

On the other hand, the increasing memory size and its raising information content created protection problems to prevent intended or unintended modifications of main memory content.

Another important step in operating systems development was the creation of the so called command language, used for the program-

---

53 - Kurzban et al. (1975) emphasize that the utilization of magnetic tapes by the monitor made necessary to limit the use of input/output devices by programmers:

"Note that this is a typical exemple of a problem caused by a solution" (pg.4)

54 - "The rapid dispersal of operating systems is a striking exemple of the influence of cooperation and communication on the information processing business. In the absence of SHARE and other user group, a few operating systems would have been developed by individual installations in splendid isolation, but the suddenly acquired ubiquity of the notion would not have been forthcoming. Aside from the users groups there are two other poyentially unifying forces in the information processing community: the professional societies and the manufacturers. By and large the pragmatic orientation of operating system development has prevented interest in them from breaking through the theoretical bias of the professional societies, and the manufacturers have been quite slow in understanding the requirement", Steel (1964), pg.27

mers to communicate with the operator and also with the computer, to give them the necessary instructions (different from programs); it was a necessary element given the separation of programmers from operators and computers resulting from previous systems software developments.

1962-1964 were years of change: multiprogramming and time-sharing systems raised new problems to software systems developers; obviously, every new operational mode had specific implications on systems software.[55,56]

All previous approaches implied that only one program was executed at a time; with multiprogramming several programs are executed concurrently, that is to say, simultaneously, but using different ressources. The final outcome is a better utilization of computer ressources, an additional reduction of idle time; this is the crucial implication for final users. With multiprogramming systems software had to face new requirements: every program must have its own protected memory area, must also have assigned priority to schedule the sequence of actions, must also perform the transfer of information from one device to another, etc.

---

55 -See in Codd(1962) a brief reference to the first multiprogramming implementations.

56 -Pyke, jr.(1967) includes a short history of the early time-sharing experiences.

Time sharing is a special case of multiprogramming with several users: a small allocation of central process unit time (a few miliseconds,generally) is distributed sequentially among users; when time is over the next allocation is assigned to the next user,even if the program is not yet finished.Unless the system is overloaded,users have the impression that they are using the system exclusively since the response time is very short.

In addition to the functions mentioned in relation with multiprogramming,software system must transfer -when allocation is over-data and instructions belonging to succesive programs- from main memory to mass storage and viceversa; has to manage communications among computer and terminals,etc.

One of the first time sharing systems was running at the MIT: the Compatible Time-Sharing System (CTSS)begun to operate on november 1963.TheARPA-SDC Time Sharing System was operational since june 1963 and the Dartmouth system started to run in 1964

The SAGE system,a military early warning system,developed in the late 1950s was the first big real time system and,by its own goals had to automatize most of its control and monitoring functions; its operating system was the origin of the most future real time systems.

The SABRE system was the first airline reservation network; it was running by 1963 and its operating system was also seminal, it was the origin of the most future work done on on-line transaction systems.

Weizer describes the situation at the end the 1950s as "very confused":

> "Most users were either developing their own operating systems or doing major rewrites of operating systems provided by the manufacturers or by other users.The industry was in general disagreements as to wether operating systems were really worth all the ressources that they consumed and if,in fact,they cost an installation much more than the installation gained from it.Most of the big scientific sys-

---

57 -On SAGE development andSystems Development Corporation role see Baum (1981).

tems had an operating system, but the smaller commercial systems (e.g. the 1400) in many cases did not, or had one that had far less capability than those on the large scientific systems.

The early `60s saw a steady increase in operating system development activity by all manufacturers of the period. IBM, Burroughs, RCA, General Electric, Philco, Honeywell, Bendix, CDC and Univac (without the Sperry) were busily engaged in developing large ammounts of systems software".[57]

In the early 1960s, in addition to the already mentioned achievements, manufacturers did some important contributions. Burroughs issued in 1963 its Master Control Program, developed in high level language and dealing with multiprogramming and multiprocessing. IBM/360 Operating System was first released in 1965 and in spite of lacking, in its first releases, several options as multiprogramming and time-sharing, its family-compatibility, the wide diffusion of the 360 and several incremental advances transformed it into the industry standard, with some specific consequences we will analyze when dealing with software-compatibility. Since then, operating systems are an almost general sub-system of all computers. Recent operating systems are designed to supervise multimode utilization of a whole computer family.

o o o

Incremental development of operating systems has covered these main functions:

a) control and management of computer ressources (main memory, peripheral memories, input/output devices, task synchronization, communications, etc.)

b) program execution (program and compiler loading, priority rules, scheduling, program library, etc.)

c) data management (file systems, security and privacy, etc.)

Obviously, every specific functioning mode requires specific functions; for that reason big operating systems are designed and in-

[58] -Weizer(1981), pg.121.

[59] -See additional information on modern IBM operating systems in Auslander et al.(1981)

plemented in modular approach.

The general trend of systems software innovation can be analyzed in three different components:

a) to raise the rate of utilization of computer ressources, reducing the idle time between succesive jobs and/or by means of a better ressource allocation;

b)to simplify operators tasks,automating part of their operations and allowing communication among programmers and operators by means of the job control language;

c) to make possible new utilization modes (time sharing,real time,etc,) concurrently with the necessary hardware advances.[60]

o o o

60 -Systems software developments -specially high level programming languages and operating systems- have induced deep transformations in the job content and in work conditions of operators and programmers;I will not analyze this important topic; see Kraft(1977) and Greenbaum(1979).

o o o

The evolution of microprocessor's software shows an interesting evolution, reproducing the main features of computer software. In the later sixties the integrated circuit industry was facing a difficult situation; the increasing complexity of integrated circuits implied a raising specificity of their applications limiting, therefore, the corresponding market; as a consequence, costs were relatively high.

The solution to this problem was flexibility, adaptability; a simple but very fruitful idea implemented by clever design: to build a software-programmable (micro) central process unit on a chip, without wired connections, using the same manufacturing techniques used to produce integrated circuits (photolithography).[61] The first microprocessor was Intel-4004, issued in 1971.[62] The new approach was the old idea on which stored programs digital computers were built; however, the approach was a new one in the components industry. Given its flexibility, the new devices could be mass-produced and one model could be suitable for several applications; when programmed, microprocessor was adpted for a specific function.

Its monolitic character, the integration on a chip, implied a jump in realibility performance. Reliability and standardization have been always powerful inducement mechanisms in electronics industry; since 1971, the development of microprocessors fostered new advances toward standarized, mass produced, highly reliable,-

---

61 -Some microprocessors include not only the central processing unit but also elements like memories (ROM, RAM, etc.), input/output control devices, etc.; they are called microprocessor systems; there are also multichip microprocessors.

62-The first 4-bit microprocessor was introduced in the US market by Intel; it was called Intel 4004 and its designer was M.E.Hoff. In 1972 was issued the first 8-bit microprocessor, Intel 8008.

monolitic and programmable microelements.[63]

The fist microprocessors were programmed in machine language because their memory was not sufficient to contain the compiler and the application program.Given that writing programs in machine language was expensive and unreliable,one technique developed later is to program a computer (a development system) using Assembler or a high level programming language,to compile the program and,finally,to transfer the machine code version to the microprocessor.

Later,the development of cheap memories with increasing storage capacity has made possible the direct utilization of high level programming languages; the compiler is resident in the microprocessor memory: Basic,Pascal,PL/M (a subset of PL/1),the C-language and other languages are now used in current applications.[64]

Hardware advances have made possible to develop operating systems for microprocessors systems and,specially,for microcomputers (generally microprocessor-based);very often they are not developed by manufacturers but by specialized firms or institutions: PC/M was developed by Digital Research,Multix by the MIT,MS-DOS by Microsoft,Unix by Bell Laboratories,etc.Generally these operating systems have been designed to be compatible or adaptable to a set of harware systems; both trends are new in the field of operating systems.

o o o

---

63 -For a detailed analysis of microprocessor development see Noyce and Hoff(1981); see also Braun and McDonald(1982)

64 -See additional information in Manuel,T.,Colin Johnson,R.(1981)

## 7. Reflections on software technological change

Since software is information to process information, software technical change is disembodied although it is not operative without suitable hardware: software is a dependent subsystem. This disembodied technical change shows a general trend: an increasing utilization of computer to use it easily and efficiently.[65] The roots of this trend are the characteristics of computers as systems capable to perform logical and arithmetical functions under the control of a special unit. In this general trend, the development of the first high level programming language and the associated compiler and the design and implementation of the first operating system appear as the crucial software innovations.

There is, however a difference in the development pattern of these two innovations. High level programming languages appeared in a complete form, like Fortran, although new languages have been developed and they are continuously evolving; this discontinuous pattern is related to their specific function which is well defined: to translate a complex language into machine language; so, language and compiler do work or they do not work but there is no mid way. On the contrary, operating system innovations have been progressive, incremental, because operating systems have been performing an increasing number of functions which have appeared progressively as necessary: hardware technical change, for example, has required new functions (related to time-sharing, real time, etc.) and software development has been responsive.

---

65 - High level programming languages, operating systems and other systems software are obvious examples of this trend. The use of a computer as "development system" to compile programs and transfer them into the micro processor is also another example.

o o o

Software applications development is generally a multidisciplinary activity requiring knowledge of mathematical algorithms[63] (statistical, optimization, control theory, etc.), a good knowledge of the concrete field which has to be modeled (chemical process, flight control, production scheduling, financial theory, etc.) and also a good modeling capacity; therefore, software applications technical change implies a strong interdependence with a large set of sciences and technologies; it is an extraordinary heterogeneous area and innovation can arise from any one of quoted fields.[64] Let us consider several cases.

Mathematical software applications are, in some way, the most self-contained. Let us consider mathematical programming: progress in this area has been related to algorithmic advances like more efficient continuous linear programming algorithms and new integer procedures like branch and bound, for exemple. Advances have also dealed with reliability, not only in terms of error theory but also developing matrix generators which reduce drastically the error rate during the data input phase.

In other fields interdependence is wider and deeper. To develop chemical process control it is necessary a precise knowledge of process dynamics under normal and extreme conditions, to generate suitable mathematical models and the corresponding numerical procedures and to programm them; in addition, to be operative, the system needs specific instrumentation, measument devices and actuators.

Robot implementation (manipulators) raises similar problems; however, there are also additional difficulties related to the mechanical nature of their moving and grasping parts: mechanical accuracy can be unsufficient for a good path and manipulating control or even can make impossible program portability.

---

66-Software and computer availability have a feedback effect on mathematics: they prompt the development of numerical and computational disciplines.

67-Rosenberg(1979) has analysed the role of interdependencies and complementarities in american technological change.

68-For a state of the art in business data processing see Champine (1980), in dedicated and control processing see Rouse(1980) and in large scale scientific computation (in Livermoore National Laboratory) see Rodrigue et al.(1980).

On the other hand business software applications, in principle, do not require neither hardware modifications nor to use additional hardware such as instrumentation[69]; the main difficulties of this kind of applications are related to real time management of big data bases

o o o

Globally, software technological change shows a high degree of plasticity; as I have already emphasized, (systems) software changes have been very often an induced reaction to changes in hardware. In addition there is one significative general rule: hardware is designed first and software follows; this plasticity derives from the immaterial and abstract character of software.

In this context it is useful to distinguish two different kind of inducemnt mechanisms, in Rosenberg's sense: problem-focusing mechanisms and solution-focusing mechanisms. Software innovation process shows frequent exemples of the first kind of inducement mechanism: the existence of idle time, different operational speeds of central processing unit, access time to different memories and input/output devices, etc.; the need for increased program reliability, for shorter response time in real time systems; the need for file protection in time-sharing and multiprogramming systems; the need to protect software from free riders, etc. are exemples of problem-focusing inducement mechanism.

Solution-focusing inducement mechanisms are also effective in software innovation process: the use of firmware to prevent software copy, specially, operating systems copy; the increasing use of computer to control through operating systems its efficiency and reliability, etc. Problem-focusing mechanisms, however, are more compulsive: although problems have also to be "discovered", solutions must to be invented, developed or implemented and, generally, there are different alternative paths so that it is often too much risky to concentrate effort on only one kind of solution.

o o o

---

69 - The trend towards the so called "electronic office", however, implies the interconection of computers with copiers, specializaed word processors, electronic mail, etc.

o o o

I have already pointed out one complementarity element between hardware and software and, therefore, between hardware and development effort: when we have analyzed systems software and high level programming languages we have emphazised that their utilization requires additional central memory and central processor time.

When the third computers generation appeared, a new approach was developed and applied: sequences of instructions controlling specific functions were stored in read-only-memories; this programs are known as microprograms, microcodes or firmware.[70] The existence of some choice among different microprograms introduces some degree of flexibility and also additional possibilities of adaptation; furthermore, their storage in read-only memories implies an additional protection element because they cannot be modified by an error, by program instructions. However, from the point of view of its development, firmware is the outcome of the same process as software;[71] nevertheless its specific storage method generates a different product, as we will see.

o o o

In-house utilization of existing modules as software building blocks, specially for large software development is an increasingly used approach; their utilization reduces development costs and if modules are reliable their use not only allows a quicker development but also contributes positively to system (software) reliability.

---

70 -Microprocessors are using increasingly firmware to implement high level programming languages (Basic, Pascal, etc); see Bernhard (1981). There are also microprocessors with firmware-operating systems; see McMinn et al. (1981).

71 -"This mouvement toward firmware, while potentially improving performance significantly, does very little to meet the fundamental needs of software design, production and maintenance" Frank (1983), pg. 63

Development and diffusion of modules through industry would be an efficient way to deal with some aspects of software bottleneck but it raises difficulties related to property rights; the existing procedures that allows to avoid these difficulties in the case of software packages (see Section 3) dont work in the case of relatively small modules; they can be easily copied so that it is only reasonable to think on public institutions and universities producing and diffusing these kind of modules. Given this constraints, modularization does not seem a powerful element prompting compatibility and standardization on an industry basis.

o o o

There are some issues related to the possibilities of substituting hardware for software as an answer to rising software costs and to increasing protection needs. As I have already mentioned this approach does not allow to avoid the crucial software development process and costs, so that hardware (or firmware) advantage is, basically, better property rights protection or more efficient computing: hardware-(or firmware)-based software is more difficult to copy and achieves better performances.

One obvious consideration is that "hardware approach" is not feasible for applications software since it implies to develop software and hardware and, generally, to use them only from time to time. Software-embedded is, certainly, a different case given its uni-task character.

An additional consideration is related to evolution: all kind of software have to evolve: this is, in principle, an obstacle for hardware-based-software. Substitution of hardware devices when errors are detected or to adapt them to new enhanced designs would be very expensive. Microcode in read-only-memory is one exception; its repetitive pattern allows reduced costs so that it is feasible to substitute one chip for another and to achieve an enhanced release.

New advances in software design and in hardware manufacture can open a new way: operating systems and compilers embedded in hardware (strictu sensu).

o o o

A related issue is also linked to the rising software costs and its labor intensive character.As a matter of fact the radical answer would be some form of automatic programming,a new way of computer utilization to use it easily and efficiently.[72] Demand inducement mechanism has been active during years;it is not only a cost reduction inducement related to persistent rising costs but also a control inducement mechanism connected to the managers need to control an activity characterized as a craft in electronic data processing management literature and, as a result,difficult to plan and to control ex-post.[73]

Although demand por automatic programming exists,up to now, there has not been a positive answer to the problem.The sources of this situation are the difficulties implied in automatic programming development,specially when the objective is a general procedure to deal with general problems; this is so because programming is not only an abstract-formal activity but an activity that requires a concrete knowledge of the field; therefore, automatic programming requires a "Knowledge base" as those used by the so called "expert systems":[74] in this context,the difficulties of developing a general knowledge base seem to be really strong; as a result it is reasonable to expect that the first automatic programming systems will be related to specialized and well structured fields.

o o o

---

72 - "Today's efforts are aimed at producing languages that will allow a simple statement about the purpose or behavior of a program to generate the entire high-level language algorithm to carry out the operations,thus eliminating the need for applications programmers",Lerner (1982),pg.1.

73 -This topic is analysed in Greenbaum(1979) and Kraft(1977).

74 -For a non technical presentation of expert-systems see Feigenbaum and MCCorduck (1983).

o o o

One crucial aspect of the software-hardware relationship is the so called (software) compatibility, that is to say, the possibility to run software developed for one kind of hardware with a different computer.

Non-compatibility among different models produced by a manufacturer means additional conversion costs if one user wants to upgrade his computing capacity or to change to a more advanced model : retraining and reprogramming costs were necessary when this non-compatibility was the rule; specially, the cost of reprogramming the applications software stock was generally very high. In the case of actual adoption, new hardware implied software obsolescence.

On the contrary, compatibility inside a computer family allows software a longer life cycle ; consequently, software costs are smaller and users can easily replace their system without software conversion costs. As it is well known, the first compatible line was launched by IBM in the mid-1960s: System 360. Several central processor units could be chosen to work with a complete line of peripherals; programs could be run on any one processing unit. Since then intra-line compatibility has been the standard approach in the industry, almost without exceptions.[75]

One of the elements of computer manufacturers strategy is to make costly and difficult users' conversion to other systems in order to strengthen their fidelity and dependence; software role is crucial to this objective which, in addition, operates as a force against standardization and compatibility. A good ilustration of this trend is Wallis' comment:

> "A manufacturer planning the provision of Fortran (say) for a new machine is obliged to ensure that the compiler supports the industry standard so that people using other machines can change systems easily. But manufacturer is not thereafter obliged to support only this industry standard; on the contrary, it is in his interest to provide enhancements to the langua-

---

75 - Although compatibility can, often, mean that software is less efficient than programs written for a specific model.

ge that are not industry-compatible, so that his customers will use these enhancement in their software and remain committed to him".[76]

On the other hand, as another strategic element, manufacturers try to facilitate at maximum the conversion to their system; the crucial tool is to eliminate the software barrier for new potential users. One classical exemple is the special software -called Liberator- offered by Honeywell in the mid-1960s with system H-200 and which allowed the adaptation of IBM software developed by the user to be runned on the Honneywell computer.

On other words, the specific characteristics of computer technology induces a bias in software technological change which can be defined as the search for asymmetric compatibility. There is one important exception to this trend: when some manufacturers have cost advantages they will, on the contrary, try to facilitate compatibility, without considering asymmetry as a necessary feature: this is the case of the so called plug compatible manufacturers.

IBM market dominance and its de facto standard has prompted the raise of plug compatible manufacturers producing systems that are compatible with IBM operating system (so that they are "software compatible"). The utilization of IBM system software involves the saving of huge software development costs and, besides, implies that there are not reprogramming costs.

**Plug compatibility has also been used by japanese firms to enter into the U.S.market: thus Fujitsu has a financial and manufacturing agreement with Amdhal and during years has been producing Amdhal components in Japan so that they have had the opportunity to assimilate the technology; Itel has a manufacturing agreement with Hitachi and Fujitsu operates also with TRW.**

---

76-Wallis (1982), pg.6

Frequently,plug compatible manufacturers use "reverse engineering" to get their product,reconstructing the design and even the manufacturing process taking as departure point IBM system; this does not mean the result will be an exact copy,among other reasons because this would imply patent problems[77];as a matter of fact,"reverse engineering" it is used as a learning procedure to achieve compatibility.In fact,extreme miniaturization is making more difficult to apply reverse engineering in integrated circuit and in computer industries but it is still widely used.

Reverse engineering would deserve deeper attention given its conspicuous diffusion in industry and the role it has played in the learning process of some countries like Japan.

In addition to prevent conversion costs and software stock obsolescence to the user,compatibility has the advantage of making upgrading feasible on an incremental basis.From manufacturers point of view it means the possibility to strengthen costumer's fidelity; it makes also possible a higher level of hardware and software standardization building common blocks or mo-

---

[77] Weil has systematized in the following fom the existing alternatives:

> "Today,IBM compatibility may mean that the hardware architecture ot the non-IBM manufactured processor or peripheral sub system is exactly the same as that of the IBM target system. In fact,the plug compatible machine is "reversed-engineered" from the design of the IBM system.Alternatively,the non-IBM hardware may be different in a number of respects from the IBM implementation,but IBM system software (DOS/370,MVS,etc.) runs on the hardware without requiring much change or adjustement.
>
> Finally,the plug-compatible hardware or software vendor (PCMs) may offer compatibility at the"user interface",even though their hardware or software designs differ from IBM's and may use the PCM's own IBM-like system software"(pg.278)

dules for all the family models; this possibility has important cost reduction implications.These advantages have some related drawbacks.To the manufacturer,compatibility implies greater design costs since he needs to deal with a complete set of hardware devices.To the user,compatibility means that software will not be optimal since it has not been developed for his specific computer. However,in spite of these drawbacks,intra-line compatibility is the standard approach in the industry,almost without exceptions.

The design of a family of compatible hardware and the related software requires a deep understanding of user needs; without this comprehension technical innovation will fail.One crucial problem is how to satisfy a wide spectrum of variable needs with a non customized design.Computer manufacturers reply has been compatibility and modularity; the possibility to link different central process units to several peripherals allows,generally,a wide number of alternative systems and this is an element of flexibility; the availability of this wide range of systems is an important element of competitive strength for manufacturers trying to have acces to large markets[78]

Software approach to the same problem is well known: to develop software packages with some flexibility so that it is possible to adapt them not only to different hardware characteristics but also to specific user needs,within limits); there is certainly a performance-flexibility tradeoff since the package is not so efficient as a customized software.

o o o

---

78 - Some manufacturers,on the contrary,produce for specific segments of the market,with a very limited range of alternative systems;Cray, for exemple,as it is well known,is specialized in big computers for scientific computing.

o o o

IBM reaction to plug compatible manufacturers has been multidimensional and complex: from price reductions to technical change, etc.[39] From our point of view is specially significative technical change oriented to modify software in order to make more difficult systems software copy; with this objective in mind IBM has implemented partially its operating system in microcode or firmware form.

As we have seen when analysing software packages, the potential public good character of software implied some specific practices directed to protect property rights; now we find the same phenomenon but connected with systems software protection, the crucial aspect of software compatibility. There are strong reasons to consider that systems software-copy-preventing is, and will be, a very powerful inducement mechanism shaping technological change in threaten firms.

The dominant firm has also reacted against plug compatible peripherals manufacturers; one typical reaction in the printers segment has been to design them so that they are no more connected to the standard channel interface, which makes possible to plug without difficulties, but to attach the printer directly to the central process unit. This is an exemple of a bias induced in complex systems tecnical change when the systems are formed by separable subsystems connected with standarized in-

---

[39] -See Weil (1982), Ch.13.

terfaces: the objective of this kind of tecnical change is to prevent the access to the system ; it is a reaction to changes in the opposite direction trying to have access to the systems as a condition for market creation.

The development of computers as modular systems with standadized connecting interfaces has had advantages for manufacturers: it has made easier modular upgading and family development etc.; on the other hand, software-hardware separation has been one condition of their characteristic flexibility. However, these two features have made possible, on one hand, plug compatible peripheral manufacturers, competing mainly with IBM and, on the other hand, traditional softaware-hardware separation is the root cause of software plug compatibility. The global result has been to facilitate the access to the computer system, breaking it into elements which can be produced and sold as individual subsystems or to open to possibility of substituting all the system ; through systems software compatibility. If actual or potential competition has access to the hardware-system this means that computer market is divided into several segments (memories, printers, etc.); this is, however, the small gate to the market; the big one is through software compatibility.

The growth of communications -in the context of new regulation rules- implies that the inducement mechanism related to having-not-having acces to the system is playing a very important role in the competitive struggle among computer-based and communications-based firms.

o o o

Stanford, may 1983

**REFERENCES**

-Armer,P.(1980),"Share.A Eulogy to Cooperative Effort",Annales of the History of Computingapril 1980.

-Aspray,W.F.(1982),"History of the Stored Program Concept", Annals of the History of Computing,oct.1982

-Auslander,M.A.,Larkin,D.C. and Scherr,A.L.(1981),"The Evolution of the MVS Operating System,IBM Journal of Research and Development,sept.1981.

-Baldwin,R.R.(1982),"Session 15: The Mythical Man-Month Revisited",Software Engineering Notes,april 1982

-Baum,C.(1981),The System Builders,System Development Corporation, Santa Monica,California,1981.

-Belady,L.A. amd Lehman,M.M. (1979),"Characteristics of Large Systems",in Wegner P.,ed.(1979).

-Bernhard,R.(1981),"More hardware means less software",IEEE Spectrum,dec.1981

-Blumenthal,D.A.,(1983),"Lifeforms,Computer Programs and the Pursuit of a Patent",Technology Review,febr-march,1983

-Boehm,B.W.,(1973),"Software and its Impact: A Quantitative Assessment",Datamation,may 1973

-Boehm,B.W.(1981 a),"Improving Software Productivity",in Productivity,an urgent priority,Compcon-IEEE,fall 1981.

-Boehm,B.W.(1981 b),Software Engineering Economics,Prentice Hall, Englewood Cliffs,1981

-Braun,E, and McDonald,S.(1982),Revolution in miniature: The History and Impact of Semiconductor Electronics,Cambridge University Press,Cambridge,1982

-Brooks,F.P.jr.(1975),The Mythical man-month (Essays on Software Engineering),Addison-Wesley Pu.,Reading,Mass.1975

-Business Week,(1980)"Missing Computer Software",sept.1,1980

-Champine,G.A.(1980),"Perspectives on Business Data Processing", Computer,nov.1980

-Codd,E.F.(1962),"Multiprogramming,in Alt,F.L. and Rubinoff,M,eds. Advances in Computers,volume 3,Academic Press,New York,1962.

-Cragon,H.G.,(1982),"The myth of hardware/software cost ratio", Computer,dec.1982

-Data Decisions (1982),"Systems Software Survey",Datamation,dec. 1982.

-Data Decisions (1983),"End Users Rate Applications Software", Datamation,march 1983.

-Denicoff,M.(1981),"Sophisticated Software:The Road to Science and Utopia",in Dertouzos,M.L. and Moses,J.eds.(1981),The Computer Age: a Twenty-Year View,The MIT Press,Cambridge,1981

-Dolotta,T.A. et al.(1976),Data Processing in 1980-1985,J.Wiley New York,1976.

-Electronics (1983),"1983 World Markets",jan.13,1983

-Feigenbaum,E.A.,Mc Corduck,P.(1983),The Fifth Generation,Addison-wesley,Reading,Mass.1983.

-Fox,J.M.(1982),Software and its Development,Prentice-Hall,Englewood Cliffs,N.J.,1982.

-Frank,R.J.,(1979),"Patent Protection and the Computing Arts", Computer,july 1979

-Frank,W.L.(1979),"The new software economics",CW,jan.1979

-Frank,W.L.(1983),Critical Issues in Software,J.Wiley,New York, 1983.

-Freeman,Ch.,(1974a),The Economics of Industrial Innovation,Penguin Books,Harmondsworth,1974

-Freeman,Ch.(1974),"Economics of Research and Development",in Spiegel-Rösing,I.and Solla Price,D.,eds.(1974),Science,Technology and Society,SAGE Publications,London,1974.

-Gillet,W.D.and Pollack,S.V.(1982),An Introduction to Engineered Software,Holt,Rinehart and Winston,New York,1982.

-Graham,A.K.(1982),"Software design: breaking the bottleneck" IEEE Spectrum,march 1982.

-Greenbaum,J.M.(1979),In the Name of Efficiency:Management Theory and Shopfloor Practice in Data Processing Work,Temple University Press,Philadelphia,1979.

-Greguras,F.M.,(1981),"Copyright primer for the information industry",in Advances in Software Technology,1981,IEEE,N.J.

-Hill,I.D. and Meek,B.L.eds.(1980),Programming Language Standardization,Ellis Horwood Ltd,West Sussex,1980.

-House,Ch.H.(1980),"Perspectives on Dedicated and Control Processing",Computer,dec,1980

-Ippolito,S.J.(1979),"Measure for Countermeasure",Datamation,febr. 1979.

-Jensen,R.W.and Tonies,Ch.C.(1979),Software Engineering,Prentice Hall,Englewood Cliffs,1979.

-Keen,P.G.W.and Gerson,E.M.(1977),"The Politics of Software Systems Design",Datamation,nov.1977.

-Kraft,Ph.(1977),Programmers and Managers,Springer Verlag,New York, 1977.

-Kurzban et al.(1975),Operating System Principles,Petrocelli-Charter,New York,1975

Larsen,G.H.,(1973),"Software: A Quantitative Assessment ot the Man in the Middle Speaks Back",Datamation,nov.1973.

-Lerner,E.J.(1982),"Automatic Programming",IEEE Spectrum,august 1982.

-Lientz,B.P.and Swanson,E.B.(1980),Software Maintenance Management,Addison-Wesley Pu.,Reading,Mass.,1980.

-Manuel,T,Colin Johnson,R.(1981),"Buying Sofware gets systems to market sooner: a special report",Electronics,april 21,1981.

-McMinn,C.et al,(1981),"Silicon operating systems standizes software",Electronics,sept.8,1981.

-Metropolis,N.et al.(1980),A History of Computing in the Twenty Century: A Collection of Essays,Academic Press,New York,1980.

-Mooers,C.N.,(1975),"Computer Software and Copyright",Computing Surveys,march 1975.

-Mossinghoff,G.J. and Lawson,W.S.,(1981),Technology Assessment & Forecast,U.S.Department of Commerce,Patent and Trademark Office,Wsahington,D.C.,1981

-National Academy of Sciences,(1979),Science and Technology.A Five Year Outlook,W.H.Freeman and Co.S.Francisco

-Noyce,R.N. and Hoff.jr.M.E.(1981),"A History of Microprocessor Development at Intel",IEEE Micro,febr.1981.

-Parker,R.W.(1965),"the SABRE Systems",Datamation,sept.1965

-Phister,jr.M.(1979),Data Processing Technology and Economics, Digital Press,Santa Monica,1979

-Pyke,jr.Th.N.(1967),"Time-Shared Computer Systems",in Alt,F.L. and Rubinoff,M.,eds.Advances in Computers,Volume 8,Academic Press,New York,1967.

-Radice,R.A(1982),"Productivity Measure in Software",in Goldberg R.and Lorin,H.(1982),The Economics of Information Processing, volume 2,J.Wiley & Son,New York,1982.

-Randell,B.(1979),"Software Engineering in 1968",4th Conference on Software Engineering,IEEE,1979

-Rogrigue,G.,et al.(1980)"Perspectives on Large Scale Scientific Computation",Computer,oct.1980.

-Rosenberg,N.(1969),"The direction of technological change: inducement mechanisms and focusing devices",Economic Development and Cultural Change,1969; included in Rosenberg,N.(1976)

-Rosenberg,N.(1976),Perspectives on Technology,Cambridge University Press,Cambridge,1976.

-Rosenberg,N.(1979),"Technological interdependencies in the American economy",Technology and Culture,jan.1979

-Rosenberg,N.(1982),Inside the Black Box: Technology and Economics Cambridge University Press,Cambridge,1982

-Rosin,R.F.(1969),"Supervisory and Monitor Systems",Computing Surveys,march 1969.

-Sammet,J.E.(1969),Programming Languages.History and Fundamentals, Pentice Hall,Englewood Cliffs,1969.

-Sammet,J.E.(1981),"An Overview of High-Level Languages",in Yovitz,M.C.ed.,Advances in Computers,volume 20,Academic Press, New York,1981.

-Simon,H.A.(1969),The Sciences of artificial,Cambridge,The MIT Press,1969.

-Solomon,L.,(1979),"IBM versus PCM's",Datamation,febr.1979.

-Steel,jr.T.B. (1964),"Operating Systems",Datamation,may,1964.

-Stoneman,P.(1976),Technological Diffusion and the Computer Revolution,Cambridge Univ.Press,Cambridge,1976

-Titli,A.et al.(1979),Analyse el commande des systemes complexes, Cepadues Eds.,Toulouse,1979

-Van Horn,E.C.(1980),"Software must evolve",in Freeman,H.and Lewis II,Ph.M.,Software Engineering,Academic Press,New York,1980.

-von Neumann,J.(1945),"First Draft of a Report on the EDVAC",in Randell,B.ed.,The Origins of Digital Computers.Selected Papers, Springer-Verlag,Berlin,1982.

-Wallis,P.J.L.(1982),Portable Programming,A Halsted Press Book, J.Wiley,New York,1982.

-Wasserman,A.I.(1978),"Towards the Engineering of Software: Problems of the 1980's",in The Oregon Report.Proceedings of the Conference on Computing in the 1980's,IEEE,1978

-Wasserman,A.I.and Gutz,S.(1982),"The Future of Programming",Communications of the ACM,march 1982

-Wasserman,A.I.et al.,(1978),"Software Engineering: the Turning Point",Computer,sept.1978.

-Wegner,P.(1980),Research Directions in Software Technology,The MIT Press,Mass.,1980

-Weil,U.(1982),Information Systems in the 80's,Prentice Hall Englewood Cliffs,1982.

-Weizer,N.,(1981),"A History of Operating Systems",Datamation, jan.1981.

-Wexelblat,R.L.,ed.(1981),History of Programming Languages, Academic Press,New York,1981.

-Zelkowitz,M.V.(1978),"Perspectives on Software Engineering",ACM Computing Surveys,june,1978

o o o