ARCHIVES AND MUSEUM INFORMATICS

TECHNICAL REPORT

ISSN 1042-1459 No.2

Collecting Software: A New Challenge for Archives & Museums

by David Bearman

Originally published as Archival Informatics Technical Report
vol.1, #2, Summer 1987
Reprinted 1990

COLLECTING SOFTWARE:
A NEW CHALLENGE FOR ARCHIVES & MUSEUMS

EXECUTIVE SUMMARY

For forty years software has been an important creative product of our society. Its intellectual, social, economic and political impact has shaped the contemporary world, yet the community of culture preserving institutions has failed to document its evolution. There is not a single archive or museum in the world devoted to software and no substantial collecting of software history has taken place in other repositories although software is being written every day which defines the way in which we work. Bureaucracies, including governments, are entirely dependent upon software to faithfully execute the policies (including laws and regulations) which they have established, yet archives, those gaurdians of bureaucratic accountability, don't retain software. Popular culture and the arts have both been transformed by software, but museums have yet to collect it. To archives and museums, software is still alien and insubstantial.

This report examines the history of software and its influences on our society and it addresses the barriers to collecting software as a cultural record. It identifies essential policy distinctions which administrators will need to consider between software collections and other collections of archives and museums. It examines the ways in which software can best be described, made available to researchers, and exhibited and it proposes a framework for a descriptive vocabulary. Further, it identifies the physical requirements and management issues associated with the retention and storage, retrieval and use, of software in cultural repositories.

An earlier draft of this report was prepared for the Computer Museum in Boston as the framework for a discussion with staff of the Smithsonian Institution and the Charles Babbage Institute on establishing a national software collecting consortium. The report, therefore, considers approaches to multi-institutional collecting issues such as documentation strategies and collection policies, cooperative acquisitions and information sharing.

The body of the report addresses five fundamental questions about such a program:

What is the domain of a software archive?
What is the mission of such a program?
What policies are required to implement a software archive?
Who are its users and what are its uses?
What procedures are needed to establish a software archive?

COLLECTING SOFTWARE
TABLE OF CONTENTS:

INTRODUCTION

The digital computer, an electronic device capable of carrying out a sequence of instructions communicated to it in a "program" which can be executed without further human intervention, had its origins in wartime applications in the 1940's. It emerged from the war as an experimental machine, evolving rapidly at universities during the late 1940's into a practical engine for numerical calculations and logical operations. Its impact on our daily lives has been so dramatic during the last thirty years that few other events in human history can be appropriately compared; and it is much too early for us to know how it may transform our civilization.

However complex and interesting the machines which have launched this revolution are, it is not they, but the instructions people have written for them to execute which are redefining the world in which we live. These instructions, or programs, written in languages which can be compiled or interpreted into machine code, are among the most exciting creative achievements if our day, yet their content, structure, cultural impact and socio-economic significance are only now becoming the subject of critical historical study. Early software masterpieces, monumental intellectual achievements of the genre, have already been lost to future study through neglect. And contemporary software concepts, rarely first embodied in widely sold general purpose systems, will likewise be lost unless the cultural history repositories of our day - the museums and archives of our contemporary society - take action soon to prevent the disappearance of this record.

In 1986, the Computer Museum, a cultural repository dedicated to collecting computers and computing artifacts, decided to explore the feasibility of a software archive. They contracted with the author to examine the nature of software and its documentation, and define what it meant to collect software as a cultural achievement. This document grew out of that project. It calls a collection of software and software related documentation a "software archive" and examines what a software archive might be. It endeavors to demonstrate not only that such a program is possible, but also that it would have substantial cultural value, and may even be necessary, to document modern political and economic institutions. It suggests that most existing museums and archives need to consider software within the scope of their

present collecting policies.

This report poses broad questions about software, about documentation efforts and about the requirements of such a program. It shows that there are no fundamental barriers to proceeding and then identifies guidelines which could be used in administering such a program.

I. THE DOMAIN OF A SOFTWARE COLLECTION

To plan for the preservation of software, we must first understand what it is and what social activity generated it. We must then examine the types of evidence these activities left behind. Finally we must consider the challenges of collecting and providing access to an adequate record of that activity. We may make the history of software itself our framework for collecting or we may take an external reference point and make the history of that area of human endeavor, an "application arena", our framework. In either case we must understand the nature of software itself.

A. SOFTWARE HISTORY AS A FRAMEWORK FOR COLLECTING

Computers are machines operated by instructions which are input to them in a form they can execute. These instructions, or programs, began to be called software relatively late in the history of the computer, to distinguish them as components of the system from the hardware, or equipment.[1] This distinction, while sufficiently clear in principle, has in fact never been absolute; the emergence of the term firmware, to denote instruction sets embedded in hardware, reflects the increasing resolution of this still gray area. For the puposes of a survey of the universe of potential software documentation issues, software must be considered to include any instruction set, however embodied, thereby forcing us to consider (although not necessarily to be subsequently collect) firmware by the original equipment manufacturer and by third parties.

Likewise, for the purposes of assessing the range of a comprehensive record of software, no purpose is served by defining an arbitrary moment

[1]My search for the origin of the term "software" ended at the Time-Life volume entitled Software which locates it as being in general use by the early 1960's. I am willing to assume that if there were a better answer, Time-Life researchers would have found it.

after which computer programs can be considered software; the potential domain for a software archive must extend to the earliest computing devices. Experimental computers, those developed prior to 1948, lacked control units and stored logic sequences and operated in part under the control of mechanically set switches. Although "programs" were written for these machines,[2] the processing was not fully controlled by the programs. Computers for which software could operate as a complete instruction set, required transfer of control and stored logic (demonstrated by the SSEC machine, 27 Jan 1948), electronic registers (demonstrated by the MADM, June 1949), a control unit and a memory size adequate to support it (demonstrated by the ACE machine in 1950, and EDVAR in 1951), and paging of memory (although not automatic, demonstrated by EDSAC 6 May 1949). All these features did not come together in practical machines until the advent of the UNIVAC I and IBM 701, the first "mass produced" computers[3], in 1952/3. Nevertheless, a number of quite important computer programs were written before 1952/3, and many important software concepts emerged from this period. In documenting the history of software, we cannot ignore these developments simply becaust the term "software" had not been coined, or because executing a particular program on these machines required a specific prior physical configuration and, sometimes, human intervention during execution.

But the symbiosis in these years between particular devices and the specific programs written to control them does have implications for the establishment of software archives. Since each of these experimental computers was unique and the methods of expressing instructions were non-

---

[2]Much important software developed on pre-1946 machines is known from the published literature, although this is inadequate for understanding the sources of certain ideas or the trial and errror involved; cf.
Adele Goldfine's, 1946 trajectory calculation program for the ENIAC which have recently been reprinted in Randell, Brian, ed.; The Origin of Digital Computers: Selected Papers, 1982
Claude Shannon's, 1938 logic program for his switching relay published in Transactions of the American Institute of Electical Engineers, or R.E. Beard's, 1942 actuarial table calculation programs published in the Journal of the Institute of Actuaries, v.71 p.193-227

[3] Mass produced is to be understood as a method of production, not an indication of volume. UNIVAC sold only fifteen of its Model 1's. IBM manufactured only 19 of its model 701 (the number sold is debated) and sold only fifteen of its Model 702.

standard, and tied closely to the physical device, these programs and the hardware on which they ran must be considered to be a piece,with responsibility for documenting both assumed by the cultural repository which acquires the hardware. The function of a software archive with respect to such early programs will be to impress upon the museum repositories that in undertaking to preserve the meaning of the artifact, they are assuming responsibility for documenting the problems which the machine was given, the way in which these problems were set, both in hardware and software, and how execution took place. There may be occasions when the machine for which such an early program was written no longer exists or is not being preserved, and, if adequate documentation exists to describe the machine itself and its functioning, the software written for it may be intelligible. With few exceptions, however, the domain of a software archive will encompass computer programs written since the mid-1950's for machines (and later operating systems and languages) which are non-unique; software for unique devices will continue to be collected along with those artifacts, by museum curators.

Software may be the product of the machine manufacturer, academics, the user, or of commercial "third parties" (who are relative late comers). Writing computer programs as an activity separate from designing and building computers, arose with the commercial distribution of computers in the early 1950's. However, when it first moved out of academic laboratories, it remained for several years largely confined to the province of the computer manufacturers, in collaboration with their (large) customers. An early example of a program of this sort was that written by UNIVAC engineers as part of the first computer sales effort - the splashy joint venture with CBS in November 1952 to predict the Presidential election results from exit polls and to launch the UNIVAC I. Collaborative relations between the customers of computers and the manufacturers, with spillover between hardware design and application, continued to be the norm in the period when each computer was dedicated to a single task. Thus IBM developed not only the software to run the Social Security system on its model 701 in 1953, but also developed (and subsequently sold) a tape processor to store the dataset.[4]

---

[4] The IBM tape unit is discussed in Rene Moreau, The Computer Comes of Age: The People, the Hardware and the Software (Cambridge MA, MIT Press, 1984). Also, see

As with many government procurements of the period, new hardware and software concepts were developed around the statement of a practical problem and multi-million dollar computers were sold to process a single application. Because each model of commercial computer during this period was sold to a small number of sites (rarely over 20), the documentation of software and hardware after the mid-1950's can be segregated. In documenting custom application systems, both the records of the design process and those of the client are important. Unfortunately for the future of a software archive, it does not appear that either the size of these early custom systems or their huge costs assured the preservation of a documentary record. Preliminary inquiries have failed to find records of several important developments.

An explosion of computer programming which laid the foundations for most concepts being refined today, took place in the late 1950's. With the development of paging (moving data between layers of the heirarchy in fixed blocks) and the evolution of drum memories and eventually disks packs (introduced by IBM as RAMAC in 1956), the computer became a sufficiently general tool to require local engineers who were capable of writing instuctions for new applications. Many of these locally developed tools were adopted at other sites either informally, like Bob Patrick's (General Motors) "monitor system" for the IBM 704 or formally, like Ray Nutt's (United Airlines) assembler for the IBM 701.[5] Such applications in themselves, together with the development of general languages for writing machine independent insructions, such as FORTRAN, LISP, ALGOL and COBOL, all of which had their origins in the late 1950's,[6] ushered in a new phase of computer program evolution, during which the concept of software as a separate entity begins to have meaning. Because a single model was available in numerous sites (sales now exceeded 1000 for successful computers),

---

Charles J. Bache, Lyle R. Johnson, John H. Palmer & Emerson Pugh, IBM's Early Computers, (Cambrige MA, MIT Press, 1986).

[5]On Patrick see Moreau, ibid.; on Nutt, see Bache, ibid.

[6]John Backus "History of Fortran I, II, III" , in Wexelblat, ed. History of Programming Languages, (NY, Academic Press, 1981); also articles on LISP, ALGOL, COBOL etc.

and because numerous applications were being run on a single machine, expertise was developed in house, and software concepts began to be developed from a community of programmers, rather than as a reflection of the manufacturers' requirements for installing a system. Because the control unit was given increasingly complex tasks to manage, with the advent of new input and output devices and deeper layers of storage, the need for operating systems was general. And because operating systems could handle physical level instructions which were repetitive, higher level languages began to evolve, each requiring a compiler for every new machine, and each giving rise to application routines and libraries in a community of specialists. By the late 1950's this activity was reflected in the evolution of institutionalized software exchanges, such as the IBM user group, SHARE.[7] Documenting the evolution of software will involve documenting these social contexts for the dissemination of knowledge of operating systems and libraries of system management routines in the transition from single processor systems to multi-processor sites with numerous I/O devices and complex communications.[8]

During the 1950's, computer programming became evolutionary in another extremely important respect - programmers built higher level concepts on lower level ones using new computing "languages". Many important programming concepts of this period were embodied in languages, so that each language came to be regarded as being most appropriate to particular kinds of problems and each provided only certain tools. Thus the history of programming languages is particularly important at this time; fortunately languages are the one area in the history of software for which a modest archive has already been established.[9] The emergence of programming

---

[7]Bache, op.cit. #4 discusses the origin of SHARE. It is worth noting that IBM archives contains "most of the published SHARE reports."

[8]Brian Kahin discusses the sociology of software distribution and the efforts to establish a new mechanism better suited to the needs of academia, in his draft report on the EDUCOM Software Initiative (November 11,1986), cited by permission of the author with the understanding that a final draft of this report will be issued by EDUCOM in the near future.

[9]Jean Sammett maintains an archive relating to languages at the IBM Federal Systems Division, in Bethesda Md.

languages marks a radical departure with major impacts for any potential software archive; programming languages are documented virtual machines and software written in them can be comprehended without the necessity of maintaining the kind of machine specific documentation required to understand software written in lower level code. The history of LISP, which was developed before John McCarthy had access to a computer which could run it, and of the FORTRAN programs written by prospective IBM 704 customers before a compiler for FORTRAN was written, provide a dramatic illustration of this independence of such virtual machines from actual computers.[10]

While operating systems and programming languages were being constructed as the base of a software revolution, computer applications were attracting substantial public attention. Following the publication of Edmund Berkeley's Giant Brains: or Machines that Think, in 1949, such journals as the Harvard Business Review were quick to focus on the potential of computers, and devoted numerous serious articles to following the utility of applications in their areas of interest.[11] Even though general purpose computing was still a modest component of the usage of computers (by far the most extensive use of computers was in military applications such as atomic energy, ballistics and eventually the space race), the potential of software as a competitive edge was clearly demonstrated by some pioneering efforts, such as the SABRE airline reservation system developed by American Airlines and IBM from 1956 to 1962. This developing public awareness of software, and of the ways in which software could be used to competitive advantage, is itself an object of documentary interest in the domain of a software archive.

While software was already a commercial asset in the early 1960's, as demonstrated by the success of Computer Sciences Corporation which was

---

[10] John McCarthy reports that since the N.E. Computation Center was not scheduled to get its IBM 704 until 1957, the design of LISP, including such central issues as how to use the 15 bit register which gave us cdr and car, were made without a computer. In the history of FORTRAN, we find sites which were awaiting the delivery of the compiler writing application code to be run when a compiler was delivered. See Wexelblat, op.cit. #6

[11] Edmund C. Berkeley, Giant Brains: or Machines that Think (NY, John Wiley & Sons, 1949). also; Editors of the Harvard Business Review, The Digital Computer: Monster or Slave (Boston, HBR, 1955)

formed in 1959 to sell contract-developed programs, the emergence of an independent software marketplace for software as a commodity required a sufficiently large population of computers of a common type to assure that multiple users would purchase or license a given product. This condition, was on the verge of being met by the IBM 1400 series when IBM pulled the rug from under its users, destroying their software investments with the introduction of the IBM 360. It was met by the 360 and its successors. Operating systems were sold as part of the computers they operated; it was no longer possible for local programmers to make or even borrow code since the number of lines lines of instructions required had grown from a modest 5,000 lines for the IBM 650 to several million lines for the IBM 3601[12] Other software tools, as they began to be developed, were leased by the manufacturers. While application software was being developed in numerous computer using organizations, it was not until the 1970's that it was commercially distributed by third parties dedicated to developing software for resale.

By 1970, one can begin to talk about a fourth period in the the history of software - the period of commercial, third party, unbundled, software development and the parallel development of a distribution system which can only exist in contast to the commercial system, that of public domain or free software exchanges. Groups of organizations with common requirements whether trade organizations, government agencies, or non-profit cultural institutions, began to develop software in consortia or developed mechan-isms to exchange software among themselves. The emergence of software as a commodity and the creation of pricing structures, markets and suppliers, is a part of the domain of a software archive.

Computers were developed to support military objectives and perform calculations at a speed which surpassed that of people, but it was not long before the computer was put to work as a routine file clerk in civilian projects, maintaining large sets of data and retrieving required information

---

[12] John Pfeiffer, The Thinking Machine (Philadelphia, J.P. Lippincott, 1962) based in large part on a TV program broadcast by CBS, oct. 26 1960, as part of the Centennial of MIT. In retrospect, given the speed at which innovations began to impact on daily life, it is startling to realize that the MIT Computation Center, the earliest computing facility at a univeristy which was not dedicated to developing new computers but using existing computers, was opened to researchers from 30 universities in June of 1957.

from them. In these functions the computer performed work which could have been given to people, but was repetitive and dull. Routine bookkeeping on a large scale was error prone; searching for files of cases was likely to result in misfiling them after the tasks were completed. The computer not only assisted in these operations, it eventually made possible file management on a vast new scale, such as that of the credit card companies or the Social Security Administration, thus shaping the way in which business was conducted by making it possible to conduct business in that way.

Until about 1970 the impact of computers on such large scale enterprises was imitative of the human processes (although, because computers cost over a million a dollars and were expensive to lease and because large staffs and special facilities were required to run them, we can assume it was cost effective). During the 1970's this changed. As computers came down in price and their performance was improved more and more processes were automated, thereby qualitatively transforming the functions which they controlled. By the end of the 1970's, computers as powerful as those used by large scale enterprises in the 1950's were available to small businesses, and they began changing the way in which they managed their internal functions. Concurrently, the larger enterprises had figured out how to coordinate a great deal of their processing from the initial component order to the final collection of receipts on the product, and this end to end automation was beginning to involve the ultimate customer, the consumer. The consumer became aware of automated inventory systems at the cash register, of automated banking transactions at the 24 hour teller machine and of large databases about himself through the "personalized" targetted mailings he received daily. The domain of a software archive encompasses documentation of such fundamental cultural effects as those of changing the scale of business and government, transforming mans sense of self[13], and altering the balance of power between information rich and information poor.

The availability of computers to small enterprises and individuals in the 1980's had immediate implications for the development of software which became evident following the release of the PDP-8. For the first time computer owners lacked the specialized staffs to develop their own software,

---

[13] Sherry Turkle, The Second Self: Computers & The Human Spirit, (NY, Simon & Schuster, 1984).

and they were numerous so they represented a potentially lucrative market. A new kind of software market, the market of software as a consumer products evolved, and found that customers were anxious to buy. The level of investment in software by users grew rapidly, creating pressure on the manufacturers to maintain stable operating environments, or downwardly compatible ones at least. While this stability benefited their software competitors, the manufacturers accepted the inevitable challenge to their software monopolies, and eventually came to enjoy the benefits of having buyers attracted to their products because of the range of third party software which was available. The transformation of software into a consumer product is another concern of the software archive. Documenting this phenomenon will require not so much the acquisition of these off-the-shelf products, as the documentation of the emerging system for their delivery and the secondary business in consumer accessible on-line data services which they are making possible.

How is it that these new products readily found customers? In effect they used existing outlets. They were reviewed in existing journals and taught in existing schools or through existing user groups. They were touted (and panned) on existing electronic bulletin boards and methods for exploiting them were published by general distribution commercial publishing houses and sold in general book stores. In the process, some of these conduits were transformed as well. One national chain of discount book stores opened separate outlets devoted to software inventories. A recent issue of Science, carried in its new product announcement section, announcements of seven new products, all of which were software based![14]

Of course, not all marketplaces were effected by the emergence of the software industry. Only one customer continued to exist for national security systems and the Federal government's program administration support systems and a small number of customers with highly specialized needs dominated such major arenas as the stock markets or commodity exchanges. In these areas, custom developed software, at increasing expense, continued to be the norm. Often it is still the case that these applications require new hardware to be designed especially for them; thus the market continues to

---

[14] Science, December 19, 1986

witness a stream of special purpose machines ranging from mini-computers to super computers with custom programming. The pressure to reduce the costs of these software applications led to the construction of numerous specialized tools for all aspects of the software development, testing and implementaion process. These tools were then used in the construction of custom solutions and in the development of commercially available software. As a consequence, the market for software workbench products, productivity tools and testing tools itself became an arena for commercial product development. Increasingly powerful tool boxes incorporating higher and higher level features including artificial intelligence applications for self checking began to be produced in commercial quantities. These same large systems also required substantial communication networks which were themselves under the control of sophisticated software systems. For these systems, software development is a vast organized effort akin to major construction projects. There are increasingly specialized roles assigned to participants and growing degrees of similarity in the tasks of designers, project managers and technicians in software projects and other large scale engineering efforts. The software archive will need to continue to document changing methods and tools in software development. As a practical matter, this may involve indentifying software efforts during thier active life and working closely with those responsible for them to assure that adequate documentation is maintained.

As noted earlier, software functions are increasingly being migrated into chips, freeing the memory of machines for more complex software driven functions and decreasing the access or execution time by precious nanoseconds. While the proliferation of software over the past fifteen years presents a challenge to those who would document it, an equally complex challenge derives from the symbiotic relationship between hardware and software. From the earliest computers forward, architects of computer systems embodied in the hardware functions which would otherwise require software instructions. As computers evolved in complexity, the governance of storage heirarchies and input/output devices required increasingly complex operating system. Over time, some specialized applications could be better performed with machines designed to satisfy applications requirements at the hardware level without the overhead of those general operating systems software functions not used by the specialized application. With the

implementation of solid state devices (printed circuit boards and silicon chips) in the place of transistors, which had themselves replaced the vacuum tubes of the earliest computers, a means was found of providing specialized instructions in hardware which had earlier been designed in software. The software archive must be alert to the source of hardware innovations in software and of software innovations in hardware, documenting special purpose computers as the source for software innovations resident on general purpose machines and visa versa.

Analysis of the history of software will always play a significant role in defining the potential scope of a software archive. Such research will, therefore, need to be provided for as an integral component of the program. In itself, such research does not define what any given archive should collect, nor how it should use the materials or who it should serve. The role of historical understanding here is to provide one set of criteria by which software might be collected. A totally different set of criteria are suggested by the view of software taken by an application based observer.

B. APPLICATIONS AS A FRAMEWORK FOR COLLECTING

While some institutions, such as a computer museum or computing department within a history museum will see collecting the documentation of software history in itself as falling within their mission, most cultural repositories should be considering collecting software archives not to document the history of software per se but to document those domains they already consider as falling within their collecting responsibility. In the terminology used by the computer industry, these institutions would be documenting automated "applications". An application is a software program which supports a specific activity - designing an aircraft, composing music, registering students for classes, or determining eligibility of applicants for insurance reimbursements. In collecting applications software, the focus of the collecting institution is on the role the software played in the activity supported by it, rather than in its contribution to fundamental concepts in the evolution of software or to the economic and social history of software in itself. Since most spheres of human activity since 1970 have been impacted by software applications, most cultural repositories which serve in any way to document human cultural achievements have some reason to consider applications software within their collecting scope.

This view of application software as a documentary record is not the same as, and indeed conflicts in some ways with, the interest among archivists in collecting "machine-readable data files". Since the early 1970's, archival programs in large governmental entities (nations, states, municipalities) professional organizations have been formed, including the Association of Public Data Users (APDU) and the International Association for Social Science Information Service and Technology (I-ASSIST). The same organizational interests and professional concerns have informed the Society of American Archivists Task Force of Automated Records and Techniques and led to the development within the Society of American Archivists of training courses and technical programs devoted to the administration of machine readable records. But these programs are not only not constructing software archives, they have, until very recently, adopted approaches which are hostile to documenting applications software for reasons associated with their need to identify formats for retaining social science data for future use. Because the reasons that data archives have been software collections are relevant to software collecting, and are discussed in more detail later in this report, they are only introduced here in order to explain an otherwise inexplicable attitude.

Machine-readable data archives have consisted of largely of the results of large scale studies (ranging from census data to satellite remote sensing data) or the contents of huge case filing systems (veterans records, accident reports or academic matriculation files). The purpose in collecting these data is to allow future generations of researchers to analyze their information content. In order to preserve the information for future use, the data archivist has, in the past, advised that it be taken out of its software implementation and recorded it in a documented, software independent, flat file tape format. In this way the information can be read back into any system the researcher is using at any time in the future and the only obsolesence concern which must be managed is to assure that the medium (tape for now) can be read by the users. Since magnetic tapes need to be copied every several years in order to preserve the information, this concern can also be addressed in the course of routine maintenance simply by copying tapes over onto newer generations of media.

The approach to saving information which has been adopted by data archives does not retain software in which the information was generated.

Nor does it document the policies and procedures which produced the information, an evidential concern of traditional archivists. When we consider that applications software embodies the instructions which are intended to produce a specific result (e.g. return an 8% interest compounded weekly on investments over $1200), the importance of retaining software as evidence that the advertised or intended consequences are, in fact, being realized, can be appreciated. In governmental settings, or with respect to regulated functions or activities for which a corporate entity might be liable, there are strong legal reasons to retain application software.

Oddly, non-financial applications software is rarely audited on a regular basis, to assure that it is, or ever was, executing policies as intended. Because software is usually being modified on an on-going basis, and because modifications may have unintended consequences, it is not sufficient to test software upon acceptance, even if it was possible to create a test data set which executed every possible part of a application (and this problem, which is an important one for software developers, has yet to be satisfactorily resolved)[15]. Output, in the form of data files which might be of interest to social science researchers and machine-readable data archives, rarely provide any indication of what was actually being done by the software program which generated it.

Finally, there are applications programs which justify retention as information in themselves. Already discussed are those which would be retained because they contribute to our understanding of the history of software (invoking new ideas in data manipulation or strategies for processing) or are creative products equal other output of literature or the arts (including musical compositions and visual presentations). Not yet introduced are software solutions which come to define a business strategy or make possible a social program. Business historians have not yet written extensively on the role which software is playing in strategic positioning of industries and within industries, but it is evident that software plays a major role in competition in a variety of contemporary business (banking,

---

[15] These problems are so serious that an entire sub-discipline in software engineering has developed to assure quality control of software and to maintain a record of all the software changes and their implications. For further information the reader should look at the literature on software configuration management.

airlines, insurance to name only the most obvious) and that the successes and failures of specific firms are directly attributable to their software.[16] Of course, we cannot ignore that software is itself a commodity, and increasingly a consumer product, which might be documented for the same reasons that other products are. Lastly, a number of public policy discussions are a direct consequence of the capabilities of software; for example, the potential for linkages between databases has created significant concerns for personal privacy (some resulting in legislation) and for national security (leading to the recent articulation of the "mosaic" theory whereby appropriately equipped software access to unclassified information sources is held to create sensitive information which might be classifiable). Such software capabilities help to define the environment in which software is received by the culture, and are, therefore, part of the background against which all applications must be understood.

---

[16] The most recent issue of PC Week, August 4, 1987, contains two lead stories which confirm this impression. "LAN Databases Help Bolster Gas-Marketing Efforts at ARCO" leads off "In the highly competitive, post-deregulation natural-gas market, Arco Oil and Gas Co. needed an edge", while the article headlined "Delta Takes Off with Revamped LAN-Based Reservation System" begins "Delta Airlines has developed a PC LAN-based computer-reservation system (CRS) that it hopes will make it the leading information-systems supplier to the nation's 30,000 independent travel agencies."

II. COLLECTING POLICY & SELECTION CRITERIA

A. CRITERIA BASED ON SOFTWARE CHARACTERISTICS

The historical synopsis in section one alerted us to the significance of different factors at different times in the history of software and suggested the connections between documenting the evolution of software and documenting a variety of general historical issues. We recognize implicitly that a software archive will need to exploit an understanding of the history of software in order to judge the importance of particular software developments, however, this tells us little about how such an understanding could be systematically used to guide collection development and interpretation. A systematic categorization of the universe will be required in order for a software archive to develop a definition of the scope which it will give to its own collecting, assuming as we must that it will be less than fully comprehensive. Each archival program must consciously review the universe of documentation and make choices about how completely, and from what perspectives, it will document any given period, place, type of software or group of people or organizations. Such choices, or collecting policies, need to be consciously made and are usually publicly articulated, for the benefit of potential donors, for other programs which are likewise collecting the history of software, and for researchers who need to know which institutions will hold collections which meet their needs. Ultimately, the principal beneficiary of the collections policy is the institution which makes the effort to state it; it can use a policy to attract support, forge an understanding of its purposes, and assess how well it is doing. Framing a software collecting policy begins with the definition of a schema which adequately depicts the universe of software in which the collection is to be a subset.

Traditionally , software has been classified somewhat uni-dimensionally, and from the perspective of the machine, according to the layer at which it operates with respect to other software - i.e., as machine code, operating system, language, application environment, application, or user interface. Using this schema, a software archive might reasonably ask what its collecting objectives were for each of these levels, and develop criteria based on priority, uniqueness, conceptual interest, market success, etc. This would, however, only give us one dimension of the history. Could we then adopt an existing, comprehensive, schema, such as that used by the ACM Computing

Reviews? It appears not. Although it may be valuable for the subsequent task of indexing the holdings of an established archive (mostly because it would have the advantage of pointing us directly into published literature), this scheme is too inconsistent about software to be reliable.* Therefore, I have proposed a variety of ways to view the universe of software, each of which supports a particular type of historical inquiry. In articulating a collecting policy based on informational (as opposed to evidential) values, the software archive should consider how its holdings would illuminate each of these dimensions, and hence how they would provide evidence for accounts of software history from these discrete perspectives:

1. Function

How did software evolve to perform different tasks? or how was it designed to meet higher level applications needs? or to fit into systems which in their sum served a larger function? Among the difficulties we will encounter here is determining what level of functions to document - controlling analog input devices? handling modem communication? editing text or managing databases? or regulating air traffic?

2. Funding Source

What kinds of capital was found for software development? How did the approaches of entrepreneurial, corporate R&D, contracted, and grant funded development efforts change?

---

* My basic point here is that no single classification can serve to guide collecting. It would be desirable to use a scheme which points to the published literature to index the acquired holdings of a repository, nevertheless, the ACM scheme has some internal inconsistencies which will have to be overcome before it can be used as an indexing scheme for acquired material. For instance, the ACM scheme provides for software under the major heading Softwar (with sub-headings for programming techniques, software engineering, programming languages and operating systems), as well as under mathematical software (a sub-heading of Mathematics of Computation, Database management - Languages (a sub-heading of Information systems), Information storage and retrieval - systems and software (a sub-heading of Information systems), Information Systems Applications - office systems and communications applictions (both sub-headings of Information Systems). In addition it provides a major heading for Computer Applications, but includes Artificial Intelligence - Languages and Software and all aspects of computer graphics, image processing and pattern recognition under Computing Methodologies.

3. Sponsor

Who paid? What patterns of interest are documentable for industry, civilian government, military, or educational institutions?

4. Computing Environment

a) Hardware

What processor(s) does it run on? What other devices are required? What interfaces are employed?

b) Commmunication Environment

How is it accessed (direct connect, timesharing, remote terminal? LAN, WAN, VAN ...?

c) Software Environment

What virtual machines? Operating Systems? Database Management Systems, Information Retrieval systems, development tools etc are in use?

[The problem faced by a software archive increases exponentially with the increasing complexity of the configuration if the object is to obtain documentation of a total systems environment.]

5. Distribution

What mechanisms for software distribution have operated, in what spheres, and with what success? How has the existence of such mechanisms shaped the software and its use?

6. Development approach

What styles of development and what kinds of processes can we document for individual, team, and broadly based colleagial development efforts? How can the evolutionary development of software through user feedback be documented?

7. Ownership Restrictions

What has the history of intellectual property control mechanisms been and how has this impacted on the character of software documentation? In particular, how have copyright, trade secret, public domain, freeware, shareware and other concepts evolved?

8. Design Theory

How have concepts in cognitive science impacted on data structures, knowledge representation and parallel processing? How have issues in software engineering influenced program structuring, information hiding, data driven approaches, etc.?

9. Business area

What industry/commercial spheres have been influenced, and how? (e.g. banking, libraries, real estate, airlines)

10. Social impact

What populations or events were effected? The challenge here is to examine impacts not just on public policy, education and the computing profession, but in terms of social history of everyday life.

This list is not, nor can any list be, an exhaustive set of valid views of the software universe. In defining what any given software archive will collect, the management of that repository must, however, ask what aspect of the history of software seeks to document, what types of questions the documentation should be able to support when the archive is mature. If, for example, the archive feels researchers should be able to ask of its holdings what kinds of software sought copyright protection and why, or what software was widely disseminated in the public domain among users of particular types of systems, and what those users could get from it, then it is essential to consciously collect from the perspective of ownership restriction, materials which might not be collected as documentation of some other view.

B. CRITERIA BASED ON FORMS OF MATERIAL

Up to this point we have spoken of software documentation as if it was uniform and identifiable. It is neither. One of the few statements we can make with certainty is that the documentation of software appropriate for collecting in an historical archive will not for the most part consist of what active users of the system considered to be its "documentation." Memoranda and correspondence written during the development process, logs of fixed

(and unfixed) bugs, market studies, RFP's, financial analyses citing competitive advantage gained from software, early designs since discarded, and the source code itself, must all be considered as part of the potential archival documentation.

Understanding of the forms of material which are likely to have been generated in the context of different types of software development and distribution histories can lead the archivist to the construction of selection criteria, freeing the program from simply passively responding to collections it is offered which fall within the bounds of its collecting policy. Only with concrete criterial and proactive collecting stance can a reasonably full documentary record be built from numerous sources, including but not limited to archival collections. The policies which dictated why the software was collected will help to define what types of documentation of software creation should be sought. If the material was obtained to illustrate a particularly elegant technical solution to a problem, then code will be necessary for its study. If market penetration due to exemplary advertising and marketting strategies is the rationale for deciding to collect a piece of software, then the advertising strategies, the TV clips and the packaging discussions are grist for the historical mills. Often a software development effort which was important from one perspective, will also have significance from another point of view. For instance, we may be interested in the first appearance of a function - screen writing languages - on a particular machine - and also in how software engineers from a number of different development contexts worked together and separately on these tools. This would dictate a search for documentation of the code, but also of documentation of the social contexts out of which the software was developed and of the patterns of communication (memo's, letters, technical reports and conference attendance) of the participants. Just as the schema of perspectives on software (developed to help define a collecting policy) can be a useful catechism for selecting decisions, it can help guide us to types of documentation which can be used to study software from any one of its perspectives. Such a schema will need to be developed fully, if we procede to establish an archive.

One additional point should be made about documentation, because it poses a serious programmatic choice for any software archive, especially in a museum or historical society: if the documentation does not exist or was never created, reconstructive research, including oral history, model con-

struction etc. can sometimes serve in place of contemporaneous documentation. Such programs are expensive and person intensive, and they are only possible while the participants are alive, nevertheless many cultural repositories choose to support active documentary reconstruction efforts. Whether this form of material will be collected must be addressed directly.

C. CRITERIA BASED ON HISTORY OF SOFTWARE

After the software archive has selected arenas for documentation (applied its collection policy and knowledge of history to the selection of candidates for documentation), and defined the kinds of records which will be required to illuminate the topics chosen (determined the value of different forms of material for the purpose), it still faces the significant task of selecting materials for retention. What criteria can it bring to this process?

The first criteria arise from outside of the target context altogether. They have been applied in the process of selecting the problem. For instance, if other records are perfectly adequate for conducting research on the subject, then limited resources argue we should look to another topic. If other repositories wish to collect certain records, and can give them adequate care, then efficiency argues that they should generally be encouraged to do so. This not very profound, but might easily be missed.

Nor is the first cirteria which we apply to the records themselves very profound. Indeed, it is a tautology: if documentation of the sort needed to support research along one or another perspective is lacking in any given case, then that case does not make a good candidate for study in that dimension. Therefore, one criterion which always applies in the selection of candidates for documentation is the value of the extant documentation for the stated purpose. If we only want to know what a clever sub-routine looked like, but have no code extant, then we cannot very well document it. If, on the other hand, we are mostly interested in the professional communication network which participated in the elaboration of the clever routine, we may be able to document a considerable amount without the code.

Selection criteria are not , however, usually so obvious to derive or simple to apply. Should a repository seek out the earliest manifestation of its focus? Or its most successful embodiments? Or manifestations which are internally interesting, coherent or rich? Should it collect an instance for each machine? or each family of machines? or each vendor? should it document the same

function in each industrial arena, or each sector of the economy? Of what interest is the criteria of nationality? Should we document each criteria for each country? Of what significance, if any, is incremental evolution? Should we, once we have decided to document a particular piece of software, collect records of all its stages and changes, all the variations and enhancements?

Nor can these criteria be considered in isolation. One repository may want to document the earliest manifestation of a new idea, even when that manifestation had little or no pratical spin-off, while another with the same collecting arena within its policy, may not. On the other hand, it might decide to collect an essentially routine intellectual development which, through good marketting or good fortune, swept the world.

Selecting criteria do not have their referent in the abstract value of the event being documented, but in the concrete value of the documentation within a specific institution and the framework of other, existing documentation. If frequently happens that the most important development is virtually undocumentable, and hence should not be collected at all, or a second case should also be selected, because it is similar and better documented. Likewise, a candidate for documentation may be less important because relatively much is already known from other published or unpublished sources.

The extent to which the development of software is evolutionary, rather than revolutionary (which will differ from one domain to another) will effect the collecting decisions of the archive. In highly evolutionary situations, in which there are influences form other developments always impacting on the making of the next product, it may be important to retain documentation of these influential elements as well as the product which is the focus of the collecting effort. The evolution may be either conceptual or market based. In developing DBase II/III, Ashton Tate broke new ground in PC database management systems but when they recently released RapidFile, a low end file manager, they were clearly responding directly to the successful marketting of IBM's PFS:file, and did not introduce any new concepts. Similarly, if we want to document the evolution of IBM's VM operating system, we must do so in the context of other IBM operating systems and their limitations, as well as in the context of the pressures exerted upon IBM by the capabilities of DEC's VMS operating system. The evolution of a single product over an extended period of time will show inventive and market advances and the

interplay of a variety of external factors. As a consequece, documentation of a software product over a period of time can expose different facts than we can see in monochronic snapshots.

Software products may be embodiments of software theories - as for instance the market for relational databases on large computers - or they may provide a stimulus for software theories - as Apple did for the user-interface research community when it invented pull down windows. Entire areas of software development, such as object oriented processing systems and parallel processing systems at the moment, and many areas of artificial intelligence until recently, exist largely as working models being sold in a marketplace which consists of other developers. The extent to which the collections in the archive are able to shed light on the theoretical roots of a particular development will be a factor in selecting documentation to be retained.

Their correctness is ultimately always measured by their appropriateness to the repository. Once each repository identifies for itself what characteristics of software will frame its collecting policy, it can apply a variety of tests to judge significance, including uniqueness, priority, impact or influence, and intellectual style and integrity. One of the major challenges of developing a plan for a software archive in any cultural repository will be to define collecting policies and selection criteria which fit smoothly into the overall goals of the institution.

D. CRITERIA BASED ON EVIDENTIARY VALUE

In effect, when we ask how an institution operated we are seeking evidentiary information. Retaining information of evidentiary value is the first obligation of an archivist to his or her institution. When making determinations about evidentiary value, the archivist does not ask about the importance of the software within the history of software, but rather about the importance of the software to an understanding of the institution or process being documented. Although the institutions and processes which are being documented are, by definition, unique and their particulars are known best by the documenting archive, it is nevertheless possible to suggest selection criteria which will be of general relevance. These criteria derive from an analysis of the process of making and implementing software the broad outlines of which will be common across institutions.

No matter how the software is acquired, a "needs analysis" of some sort will have been undertaken. In that process, the enfranchised actors within the institution expressed their requirements and dictated the kinds of functions which the software was to perform. Nowhere else will the archivist find as explicit a statement of the programmatic purpose of the software, nor better evidence against which to compare its actual performance.

Between the needs analysis and the emergence of written code lie dozens of representations of the system, ranging from data dictionaries, information flow diagrams, systems architectures and database architectures to permission tables, validation lists, operating procedures and other implementation controls. Like the many kinds of drawings used by architects and contractors in the construction of a building, these are at the same time better evidence of the nature of the system than the code, and potentially misleading (since it might not actually have functioned in the way it was designed).

Of course, there is the software code itself, but there are many reasons to be careful about relying overly on the code. Even if running software on machines which will be available in the future were not extremely problematic, the archivist needs to recognize that application code is only a tiny portion of the overall software required to run the system, and while it may appear most important for evidentiary reasons, isolated application code may not answer precisely those questions which arise most often in litigation requiring evidentiary documentation, such as "how did the list sort?" or "how could a particular calculation have failed?".

Finally, implementation effects what software does and how it performs. The ability of a software product to report on the status of a claim online in real time does not imply an implementation in which real time claim reports were available to staff. Even a stated requirement that the data in particular fields be value checked and software which permits such data entry validation, is not sufficient to document that the values were checked. Only procedural evidence can ultimately establish how the system was implemented.

III. EXTANT DOCUMENTATION & ITS SOURCES

A. POTENTIAL REPOSITORIES

To collect it, we must first establish where it is. Where is software documentation in the broad sense in which we are using the term?

To begin with, it is not primarily in the collections of other cultural repositories. The existence of other repositories interested in or already pursuing software archiving would be welcome; we know from the size of the universe that no one archive can be comprehensive. However, a telephone survey of archivists at a few computer companies, high-technology firms, universities, professional associations and governments which was conducted from March to May 1987 to determine whether they collected such material, and whether they thought the records of software still existed in the institutions for which they were responsible, exposed that they currently are not collecting this material.[17] Though incomplete, this survey exposed some salient facts about the enterprise.

As assumed, no one seems to have formed a software archive nor is anyone collecting software documentation, except as an inadvertent bi-product of institutional archives, with the noteworthy exception of one computer industry sponsored historical center in the United States and a similar effort just formed in the U.K.[18]. Even organizations such as SHARE and the Federal Software Exchange, which were formed for the purpose of "archiving" software in the data processing sense (that is keeping it for distribution during its useful life), have not retained their "back stock". A large number of such distribution programs still exist, however, and those which do (SERAPHIM, SIMTEL, EDUCOM, the University of Waterloo, SHARE, FSE etc.) would probably be very conducive to depositing their materials.[19]

---

[17] The survey, conducted by the author for the Computer Museum, was informal, but nevertheless revealing.

[18] The Charles Babbage Institute, 103 Walter Library, University of Minnesota, Mineapolis, MN 55455 and the National Computer Archive, Department of Science & Technology Policy, University of Manchester, Machester M13-9PL

[19] The Simtel 20 file server at the U.S. Army Base at White Sands, NM has a large library of public domain and user supported software which was originally built at MIT,

The survey discovered that the few repositories which are sophisticated in collecting machine readable materials, have focussed their attention exclusively on how to get the data out of software specific environment and formats in order to *dispose of the software*. The only exception to this practice to date has been in small pilot studies of DBMS and electronic mail environments where the question has been posed whether the documentation of such systems will require retention of the software.[20] Even in these experimental settings, however, the focus has been on how to avoid keeping the software (and being dependent upon it). Therefore we need to make a radical distinction between a machine readable data archive and the concept of a software archive or collection.

The concept of collecting software for historical research purposes had not occurred to the archivists surveyed; perhaps, in part, because no one ever asks for such documentation! For a variety of reasons best traced to the influence of social science data archivists over the development of machine readable archives guidelines, American archivists do not see software as having any evidential significance, and thus do not collect it as an aspect of documenting the ways in which their organizations worked.

While technological barriers have not been the reason why this material has not been collected, they would have become a reason immediately had the respondents given the matter any attention. The survey revealed that archivists assumed that executable software would be a necessary component of a software archive, if not its only holdings, and had ruled out collecting software documentation because they couldn't imagine how an archives could maintain facilities to run any software which might be acquired. When pressed to state precisely what kinds of research would require having the code and machines on which to execute it, no one was able to come up with a convincing illustration.

Once introduced to the concept of software archives, especially if such collections could be maintained without requiring the repository to become a

---

including libraries for a large variety of operating systems (especially UNIX, Ada and CP/M) and is available to users of Arpanet (or Bitnet which accesses Arpanet).

[20] John McDonald, "The New National Archives of Canada and Electronic Records, Archival Informatics Newsletter vol.1*2, Summer 1987, p.14-15

scientific endeavor[21]. The accepted literature on methods of archiving machine readable files, on archives automation and on the challenges of documenting database management and electronic mail environments, all view the software environment in which such new systems are implemented as a barrier to archives, rather than as a potential subject for documenting[22].

B. SOURCES

The process of developing, manufacturing, distributing and selling software is not fundamentally different from other research and development and marketing processes and can be expected to generate similar types of records except in a few respects, noted in more detail below. The best assessment of the significance and utility of different types of records, in the context of the place they played within the R&D process, is presented in the volume by Haas, Samuels & Simmons[23]. Unfortunately, that otherwise exemplary study provides no guidance regarding software, and little with respect to processes driven by software. When we consider that most of the new tools acquired by modern R&D laboratories are either software or software based integrated systems, the importance of providing better guidance is clear. In addition, archivists concerned exclusively with the preservation of data compiled by computers have made significant progress in the past decade in establishing standard methods for archiving of machine-readable data files. Their guidance asserts that the data should be rendered software independent by rewriting it to sequential data tapes documented by fixed field code books. This practice, while preserving the data for future social scientific analysis, has the potential of seriously impacting on our ability to understand the meaning of the data to those who

---

[21] Joan K. Hass, Helen W. Samuels & Barbara T. Simmons, Appraising the Records of Modern Science and Technology: A Guide (Cambridge, Mass., MIT, 1984)

[22] Hedstrom, Margaret L; Archives & Manuscripts: Machine-Readable Records, SAA Basic Manual Series (Chicago, SAA,1984). See also, Thomas E. Brown & William A. Reader "The Archival Management of Machine-readable Records from Database Management Systems: A Technical Leaflet", Archival Informatics Newsletter, v.1*1, Spring 1987 p.9-12

[23] op.cit., *21

operational facility for obsolete computing systems, there was considerable interest in the idea. Many of the interviewees requested copies of the final report and without prompting suggested that their institutions would be interested in pursuing the matter further. Most of the repositories contacted were, of course, natural components of a software archiving consortium or reporting network.

The absence of a community of scholars needs to be addressed as the mission of the software archive is articulated. The interest of other repositories in potentially collecting software needs to be exploited as a conscious element of the collecting strategy. And the distinction between the functions of the software archive and those of data archives needs to be made a clear focus of future presentations of the concept.

In discussing of the kinds of materials which exist in their institutions (as a postscript to explaining that there was no software archive), archivists identified numerous forms of material which would be invaluable as part of a software archive. This suggests first, that substantial documentation surrounding the history of software still exists, and second, that those responsible for it may not recognize it as forming a potential part of a software archive. However pleased we may be that some material has survived, both parts of the response represent a threat to the enterprise. Even archivists intuitively regard a software archive as consisting of code, and are therefore likely to discard other documentation. The AFIPS brochure "Preserving Computer Related Source Material" could help to raise awareness among creators of the documentation, but getting archivists to preserve appropriate source materials will depend on clearly articulating the mission of the software archive and actively promoting the concept. Advocates will need to plant the notion that the history of software is a broad-based research arena in appropriate places in archival literature. Even such sources as Appraising the Records of Modern Science and Technology: A Guide, though only two years old and written by potential participants in the software archive movement, barely refer to computer software and don't suggest the value of documenting it as subject rather than object of the

compiled it (ie. how they were able to use it in its native implementation). It is also easily misread by archivists without any technical training to recommend that they dispose of software and software related documentation, which could reduce the universe of such documentation.

Organizations which take part in collectively building the documentation of software should develop a shared strategy for defining the universe of documentation and determining collecting priorities. While such a strategy will be useful in building collections, it is not the same thing as the tactics for building the software archives. Rather it is a repertoire of tactics for analyzing the field of software history. It does not begin with records, and certainly not with surveying, nor is its principal intent to secure the deposit of records in a dedicated software archive, though it aims to assure that appropriate documentation does get accessioned by some repository, and that in the aggregate, this documentation and information already available from other sources adequately documents the history of software.

A documentation strategy begins with historical research which establishes what the significant features of an historical landscape are. Just what the problem domain includes, can vary from a specific intellectual feature (a documentation strategy for artificial intelligence, such as is dicussed below) to a general social effect (like the intelligent consumer product).

The initial questions are posed from two perspectives: what happened, and what types of documentation would have been generated by what happened. In the case of the intelligent consumer product, microprocessor chips capable of holding small instruction sets, such as the options available for a dishwasher or burglar alarm, became sufficientlly inexpensive and displays became sufficiently reliable at about the same time that they were associated somewhat romantically in the minds of consumers with high-technology which was perceived as valuable. The result was explosive. The first tactic for documenting what happened, and where, is to take a snapshot of the ephemeral consumer product literature and associated trade publications every two months for five years. What would be added by other forms of documentation? Perhaps the flavor. If so, interviews with industry executives would seem an appropriate second order documentation tactic.

The documentation strategy establishes what needs to be documented

and, in principle, how it might be documented, but part of its repertoire will necessarily include a survey, at least from reference sources, of existing records. The operating assumption here is that most concepts can be documented in a wide variety of different ways. There is rarely one single piece of information which is critical to understanding the past, and if there were, we could not know it was there before finding it anyway. Instead there is information which is known to a variety of actors, organizations, and associations and there are records which range form discarded audit trails and drafts of grant proposals to tax returns and annual reports.

1. Actors

After compiling lists of individuals who played significant roles in the area under study, the archive could assist them to place their professional papers in appropriate archival custody. To assist in the education of individuals contacted, other history of science centers have used a brochure for scientists originally developed by the American Institute of Physics to help to explain why and how to arrange for personal papers to be deposited in an archive. This brochure, or a similar one focussed on software documentation issues, would be a valuable tool for explaining to potential donors why their records are of interest and what they can do to assist.

Institutions are actors too, and while it may not be desirable to acquire their records in many cases, dedicated software archive programs, such as might be formed by the Computer Museum, the Charles Babbage Institute and others, could offer assistance to software houses, computer firms, universities and research centers, in developing corporate archives for their own records. These dedicated archives should avoid seeking to acquire such records for themselves, except as repositories of last resort, for reasons of self-preservation and politics. The records of a large software corporation or computer company are simply beyond the capacity of a Computer Museum, Babbage Institute, or even the Smithsonian Institution. The costs of entering into such arrangements, even with a very tiny firm of great importance, are that it is nearly impossible to ever back away from the relationship, even if the firm succeeds and grows beyond all expectations.

The records of professional organizations may prove more interesting than they sound. However, the disadvantages of corporate archival relations

in the for-profit sphere are magnified among non-profits. One significant exception is academia. One of the most promising areas of pay-off for the development of software archives is within university archives which could focus attention on software development and use in higher education. By documenting their own backyards university archivists could preserve much of the valuable record of software history.

A concerted effort to track down the records of the many informal and transient committees and groups, such as standards organizations and industry forums, which have played significant roles in the development of software is called for. There is no other way of acquiring this material besides systematically following all leads and tracing whether it exists. Models for such survey projects are available.[24]

2. Products

Searching for documentation surrounding a product can be a useful way to define the universe of documentation. Certain languages, specific packages, particular algorithms, have an influence on an area of computing at one time. These pieces of code and logic may have been developed by individuals whose work wouldn't otherwise be documented except that they were members of a team, or participants in a conference, or founders of a bulletin board, where the target product evolved. Often by beginning at the product end we can identify forms of documentary evidence and subjects of documentation which would otherwise escape us.

Software development does not differ greatly from other creative enterprises. It is esssentially an authoring process, whether done by an individual or a team. Obviously if a team is involved, more explicit outlines of the product, functional decompositions of its modules, and formal definitions of its interfaces, can be expected to have been produced prior to and during the process. Often systems written by individuals will have such documentation created for them only after the fact. For historical research, contemporaneous documentation is much more interesting, since it reveals changing

---

[24] Fleckner, John; "Archives & Manuscripts: Surveys", SAA Basic Manual Series, (Chicago, SAA, 1977)

assumptions, expectations and approaches. As with literary products, more can be made of drafts and discarded code than they deserve. Sometimes such abortive efforts and early stabs are interesting, but except in the case of the extraordinary product and exceptional creative genius, these materials will never be used by researchers and, indeed, little can be learned from the gropping progress towards a finished product.

Software makes generalized machines behave in specialized ways. But there are two ways to skin this cat; the other is to design the hardware to behave in the desired fashion. All software is, to some extent, making use of particular facilities of the machine environment (some of which may derive from lower level software) in which it is running. Ever since the very earliest machines, the engineers who construct the hardware and those who write the software have been separate; one of the most important and interesting issues in the history of computing is the interaction between them. It is truly a two way interaction with some functions migrating from hardware into software (typically to make them more flexible and adaptable) and others migrating from software into hardware (usually to optimize performance).

These adjustments generally take place in the development process, as part of the testing and tuning of systems, and over the course a system's life-cycle, with new releases of both hardware and software. Documentation is likely to be found in internal memoranda and progress reports; indeed without these, identifying the causes of changes in code from one iteration to the next would be difficult. Alpha and beta test documentation, will be explicit about the reasons for these changes (in the case of beta sites, the needs of users will be concretely stated here, if not in the design documents). The minutes and petitions of users groups, if these can be found, will also help explain both interactions and unilateral adjustments to hardware or software. Published reviews, benchmarks, post-bidding analysis in corporate offices which lose a major contract, and other critical assessments, also have an impact, and are excellent documentation to have in conjunction with the altered product.

3. Social and Intellectual Relations of Software

A large number of the subjects of the universe of documentation we are

seeking are not concrete entities at all, but patterns, networks of interpersonal relations or ideas. The record of influence can be every bit as physical as the object code of a system, but the subject itself is intangible. What kinds of strategies can be employed to expose documentation of amorphous things ? When seeking documentation of subjective states of mind, as we often are, it may be essential to resort to awards, rewards, and competitions to smoke out those individuals who identify with a particular belief, or participated in a debate. Gimmicks, like competitions, story telling contests, and recognition to donors can be used to attract items or classes of items to the archives. For example, in 1985-86, the Computer Museum conducted a very successful "earliest microcomputer contest", which resulted in its acquisition of evidence of this special class of developments and to a very well received exhibit[25]. It also rewrote the history of micro-computers by smoking out large numbers of individuals whose independent work was submitted for the contest but was previously unkown to historians.

The interaction between research on the historical structure of an area of potential documentary interest, and of surveys of documentation in the field (identifying the published materials, including histories, and the existing repositories of archival records), will create an further understanding among staff of a program which collects software of the nature of informational lacunae. As materials are identified for deposit in the software archives or elsewhere, the patchwork will be filled in and emphasis in the collecting endeavor will shift. Maintaining a strategic orientation, in which the objectives of "adequate documentation" have been articulated in advance and remain in place throughout the effort and against which particular collecting succeses and failures may be measured, is important both to the staff and to outside researchers. In the mid-1970's, Roy McCloed (then at the University of London) conducted a survey and documentation project on British science which reported both on where documentation was located, as well as what had been sought[26]. It even documented explicitly what documentation

---

25 The Computer Museum Report, vol. 17, Fall 1986 "The Catalog of Personal Computers" discusses the project and its reults.

26 MacLeod, Roy; Archives of British Men of Science, 1972

which had been sought had not been turned up in which places and what documentation was reported (and by whom) to have been destroyed. In retrospect it is clear that the negative reporting of the British Men of Science archival survey project was every bit as important to researchers as the positive reporting and that the combination of such an on-going account of documentation status and a clear sense of what constitutes adequate documentation is a mechanism for steering a documentations strategy and for focusing efforts where they are most needed.

More than most products of our society, software has been influenced by its own seedbed, the university teaching environment. A number of significant computing languages have their source in instructional needs (BASIC, LISP) and, because early computing was born in universities, so do many basic design concepts. Computing remains an industry which relies heavily on academics to forge new concepts. Immense programs are operated by each of the major computing manufacturers to give computers to universities and the Federal government is relying on super-computer centers associated with major research universities to give the United States the lead in the next generation of computers. Records of grants by industry and government to academics, and reports on their findings, will play an important role in documenting the history of software. Fortunately, many universities will be retaining much of this record as a routine part of their archival programs. Unfortunately, few of these programs were founded prior to 1970, and they have much to collect. Hopefully, alerting archivist to the value of these records for the history of software will increase their likelihood of retention.

To appreciate the social relations of software, we must understand what it means that software may be a commodity. While, ownership rights in software were until recently poorly protected by each of the methods available (copyright, patents, code hiding and trade secrets), any method of protecting ownership involves making and keeping records in the offices of the corporate counsel. Now that copyright and patent protection is becoming stronger, we can expect to see greater use of these methods, and greater historical interest in the files of patent attorneys and the copyright office. The requirements of the trade secrets law are such that those who use this as a method of protection have had to construct files demonstrating how they invented or developed a technique which gave them a competitive

advantage. These records, also found in corporate legal files, will be of future interest to historians.

Since software first emerged as a commodity it has been the subject of a massive ephemeral literature (in print and audio-visual formats) of advertisements, brochures, catalogs, demonstration disks, and endorsements. If the history of other aspects of our industrial society, such as the flowering of the machine age in the second half of the nineteenth century, are any indication, historians will find these ephemera (if they find them at all) to be extremely valuable evidence. Those of us who live with this stuff crossing our desks every day know that we are talking about immense volumes. Strategies for capturing this quantity of ephemeral literature need to be developed, whether they are cooperative collecting along divided lines of responsibility or random or periodic sampling.

Alexis de Tocqueville observed that American's organize for any and all purposes, and they still do. There are hundreds of organizations of persons involved in developing and using software. As formal mechanisms for communication and as part of the phenomenon themselves, documentation of these organizations will be critical to understanding the history of software. Some of these organizations, like the Institute of Electrical and Electronic Engineers (IEEE) for instance have already formed archives. Others have elected to deposit in existing archival repositories; the American Federation of Information Processing Societies (AFIPS), for instance, deposits its papers at the Charles Babbage Institute. Still others are exploring such arrangements; the American Society for Information Science (ASIS) is in negotiation with two potential repositories. Membership organization records are often thought to be barren, but committees of the Association for Computing Machinery (ACM), the American National Standards Institute (ANSI), the IEEE, and other organizations have played critical roles in the development of standards which shape software, and provided for training and professsional education. In addition they have testified before Congress, taken part in international conferences and trade delegations, and sponsored software exchanges among members and even the development of specialized software to meet shared needs.

IV. THE MISSION OF A SOFTWARE ARCHIVE OR MUSEUM

A. DISTINCTIONS BETWEEN A SOFTWARE ARCHIVE, SOFTWARE LIBRARY & ARCHIVE OF SOFTWARE

The first step in establishing a software collection, assuming questions about its feasibility have been satisfactorily answered, is to define precisely what its purposes are, and are not. It may be useful, because we are defining a new kind of cultural institution, to contrast it to existing institutions before attempting an independent definition of its mission.

A software collection in an archive or museum differs from a software library in that its primary aim is to document the history of software or the manner in which the parent organization employed software, while that of the software library (even if, following data processing usage, it calls itself an "archive") is to provide software for the short term use of its clientele[27]. As such the software archive supports research about software, not research using software. Obviously, the products which are acquired by a software library, for use by clientele of that institution for the purposes which the software was intended to serve, and which are likely to be among the most widely used and "popular" published software packages, are also part of the universe of a software archive.. The obverse, however, is not true: software documentation acquired by software archives, including documentation of developments which were not commercially packaged and of aspects of the environment of software other than its code, are not likely to be acquired by software libraries.

It is useful in addition to differentiate between two classes of software archives. A software archive may either be a distinct historical collection or a collection within an historical repository, or it may be an integrated component of an institutional archive, documenting how that institution used software, and thereby providing evidence of the functioning of the organization. In the latter case, evidential reasons may be the primary, or even sole, reason why an archival program retains software. Indeed, I argue strongly that corporate archives should retain software documentation for such evidential reasons.

---

[27] Robertson, Steven; Public Microcomputing: facilities and usage in public libraries, (Studio City, Ca; Pacific Information, 1986).

In contrast, the focus of the software archive as an historical collection is on exemplars of a creative genre, rather than on evidence of the activity of an organization or person. This kind of software archive resembles a literary, art or music collection, and like them it must collect not only specific examples of software but also materials which provide evidence of the way in which software was created, distributed and used. In this way it can support research on software itself, and on the contexts of software development and use. Just as we cannot understand art, without understanding the world of art patrons, art dealers and artists, we cannot understand software without documentation of its context of development, dissemination and use. The software archive will not consist of software products alone, without any documentation of process, and indeed might exist without any operational software code! A software archive can no more consist exclusively of code than a music collection could consist only of scores. It will necessarily hold a wide variety of forms of material including the correspondence of software developers, the business plans of sponsors, and the financial agreements of copyright agents. A software museum will also contain some examples of earlier computers and operating environments so that the software can be played to an audience which needs to know what it "felt like." Of course, the museum aspect of the software archive will also benefit from the range of forms of material, permitting exhibits to be developed around themes ranging from the sponsorship of software development to the nature of software advertising. The software library, most likely, would be interested only in holding the documentation required to "use" the software, for its original purpose. Therefore, the software archive will not be an archive of software, per se, but of documentation of the entire process of developing, implementing and exploiting software solutions.

Even though a software archive or museum will seek to further our understanding of the history of software through interpretative exhibitions and publication, it need not display software in an operating system to do this. Although exhibition purposes are best served when we can show as well as tell; an exhibit designer can teach much about software even if it cannot be demonstrated. Just as an historian of music can appreciate a score, without hearing it played, or especially without hearing it played on period instruments, a researcher in this field will be able to understand software and illustrate for others what it represents, without necessarily playing it

back on the machine for which it was written. Therefore, even for exhibit purposes, the availability of hardware and software environments appropriate for running a piece of software should not be considered a prerequisite for collecting it, although it should be taken into account in evaluating the exhibition value of the item.

In this respect, also, a software museum or software archive differs fundamentally from a software library, which has as its sole purpose the provision of software for use. Software which is not of use, would not be collected by libraries, nor should it be. But the software archive may acquire it, as well as isolated modules, routines or even sub-routines, not in full systems, since the novelty of the software, or its distinguishing features, may well be in a very minor component of the overall system. The archive or museum, therefore, may not be overly concerned about whether a compiler exists, or whether the source code itself can even be found, if other documentation makes it clear what the nature of the software innovation was, and supports a variety of types of scholarly queries. As a consequence of being machine independent, the software museum or archive can collect software developed in a wide variety of specialized contexts - such as navigation systems, energy management, robotic control, etc. While a software library must restrict itself to general purpose software, usually written of the kind of small, general purpose, computers it can afford to maintain for its users.

B. SCOPE OF THE SOFTWARE ARCHIVE

The archives of organizations in the software business, and of institutions which contract for or develop software to conduct of their business (most larger firms these days), will acquire a software archive by"retaining", rather than "collecting", materials. For them, the scope of the archive is not an issue, but for any repository involved in collecting software documentation other than as evidence of organizational activity, determining the scope of its collecting activity will be a major feature of the definition of its mission.

The decision to document the history of software in itself establishes a problemmatic goal. Unless we accept that no single archive, or small group of institutions, could achieve it, any more than they could document the history of government, we are certain to be frustrated. Organizations willing to collect software should institute a mechanism to share information with

each other about their respective holdings, and if, trust holds, about the contacts they are making[28]. Competition is not pre-ordained. The interests of different institutions will skew some towards the financial impacts, some towards the use of software in a sphere of activity (such as communications, chemistry, eductation, or air and space R&D),[29] and some toward internal developments in software concepts. Any comprehensive long term documentation strategy will necessarily require numerous organizations with such distinctive perspectives to cooperate.

---

[28]The Charles Babbage Institute has recently circulated such a documentation strategy for consideration according to Larry Hackman and Joan Warnow-Blewett, "The Documentation Strategy Process: A Model and Case Study", American Archivist, vol.50,*1, Winter 1987, p.12-47

[29]These examples were selected because some collecting has already taken place at repositories which are naturally affiliated with communication (Bell Labs archive), automobiles (the Henry Ford archive), and aerospace (the National Air and Space Museum).

V. USERS AND USES

A. USERS

1) Historians

Regardless of how an archive for the history of software presents itself or what its collecting criteria emphasize, it will be of value to historians of science and technology and other academics, such as psychologists and philosophers, interested in the historical evolution of software embodied concepts. If the collections policies permit acquisition of software which was important to particular industries or sectors, specialist historians will, in time, also be attracted to conduct research in the archive.

2) Developers and their lawyers

If the software archive makes unique design concepts a focus of its collecting, and in some way indexes its holdings to reflect such conceptual linkages, it would be of great benefit to lawyers representing software developers and to software engineers themselves. While the utility of re-usable code is still a matter of debate within the software community, the potential value of studying previous implementations of common underlying concepts is self-evident. This would be especially true if the holdings included software of recent vintage. Reciprocally, the holdings of patent attorneys and others representing software developers are also important sources for documenting the history of software.

3) General Public

In theory, the software archive could be a facility for non-scholarly, inquiry if some or all of the software collected could be run on either the devices for which it was designed or on systems emulating those devices. In practice, however, providing for casual inquiry may prove more complex than mounting an exhibit, since few if any software systems yet designed can be said to be accessible to naive users without substantial knowledge, on-line help and even pre-programming, which would have to be supplied by the curator. One exception to this rule is software documentation in the form of films, as the Computer Museum has shown in its collection of computer assisted animation movies from SIGGRAPH and elsewhere, which document the capabilities of the software whch generated them, and in the acquisition of movies showing how software operated on machines for which it was initially designed. Another exception may be demonstration disks and

educational software products intended for consumers, both of which are becoming more common, but still represent only a tiny corner of the market.

B. USE

It seems, therefore, that the software archive must be viewed as a research facility above all else, and that its support will have to come from sources other than general visitors. However, with the exception of a tiny community of historians of computing, there is not yet any research interest in software, so the creation of a user community and the collecting of a software archive will need to go hand in hand. This is particularly important if other institutions, especially universities, are to be encouraged to collect software and its history and if scholarly foundations are to support part of the costs of building historical collections. On the other hand, reasons for retaining software documentation for evidential purposes abound, and the use of such records, especially in legal disputes, will increase over the coming years as a growing number of functions are performed by liable organizations under software control. Software archiving may find more uses in the corporate environment in the short term than in academia.

Nevertheless, one concern of any software archive program ought to be to assist in building up a scholarly research community. Among the (time-honored) activities which contribute to this end, many of which are already being conducted by the Charles Babbage Institute, are:

- Publication of a regular report on archival developments, including such new holdings deposited in Institutions throughout the world (such a column could appear in the CBI Newsletter, or a general publication read by historians of science and technology).
- Employing recent graduates of history of science and technology programs in projects of the software archives, or employing graduate students to encourage dissertations using software archives resources. This could possibly include the provision of research grants to post-doctoral students.
- Holding talks, seminars, or conferences on topics in software history, and possibly publishing proceedings of such conferences.
- Publishing bibliographies.
- Seeking funding from corporations with software to finance student help in organizing and describing their holdings. Seeking to encourage university R&D projects involved in developing new software to document their

initiatives.

It is in the interest of repositories which might collect the history of software, to have the field of software history recognized by the National Science Foundation and the National Endowment for the Humanities as a developing discipline, with practitioners to whom grants can be made. In addition support, even if very modest, for the implementation of a pilot program should be sought from the NEH and NHPRC, in order to give the program visibility and legitimacy.

VI. PLANNING FRAMEWORK

A. A MODEL

How then, should a cultural history repository, go about establishing a software archive?

First, the decision to form a software archive should derive directly from the goals and purposes of the host institution and the criteria for acquisition should fit into the collection development policies of the host. Either the institution must have the commitment to documenting modern cultural history, and a concommitant ability to supporting research (because a software collection will contribute less to exhibition and public programs than many other collections), or it must have the obligation to preserve an evidential record which is embodied in software.

If the archive is to "collect", rather than schedule, software, it will need to define the criteria it will use to evaluate such gifts. In effect, these criteria are identical to those it would employ in any other self-conscious collecting effort, except that they will be extremely difficult to communicate to potential donors who will tend to equate the holdings of a software archive with programming code. Defining such a collecting effort involves identifying either products or persons (including corporations) which exemplify a "type" being sought. Approaching potential donors with a clear definition of the place which the software they possess occupies in the documentation strategy of the archive will also be an incentive to them to donate the desired materials.

Defining the significant developments and actors in any given plane of the matrix of perspectives on the history of software described in Section IIA, is a substantial research effort. The staff of a software archive might use specific exhibitions, funded research projects, special publications or probes of areas of corporate activity, to frame software collecting objectives in specific arenas. The best model for employing such special projects in the achievement of an overall documentation strategy, is the very successful Center for the History of Physics at the American Institute of Physics in New York.[30] The Center, founded in 1965 at the end of a four year survey pro-

---

30. Joan Warnow-Blewett, "Documentation Strategy Process: A Case Study", American Archivist, vol 50 #1, Winter 1987, p.29-47

ject funded by NSF, both collects materials relating to the history of physics and encourages other organizations to do so. It assists in identifying materials which should be preserved and maintains a catalog of materials housed in repositories throughout the world. Every several years the Center has launched a study of some special arena for physics research - nuclear physics, astrophysics, solid state physics, physics in national laboratories, and laser physics - with support from oustide funding agencies. These studies have served to identify important historical developments, locate existing documentation and provide information about its contents, alert collecting organizations about lacunae in collected documentation, and to support a growing community of scholars, many of who served post-doctoral years as project staff, who further interpret the field. They have also had the important secondary effect of making potential donors aware that they possess materials of interest to historical repositories and of helping the repositories to define criterial for asessing such evidence if it is offered. The full-time archivist and full-time historian on the staff have had their ranks augmented by project oriented staff throughout the life of the Center, and the project oriented staff have been reponsible for numerous publications and at least one major travelling exhibit which have enhanced the visibility of the repository. Historians of science as well as practicing scientist have proved very supportive and the Center has also made some fundamental contributions to archival practice.

The Center has found that the most interesting documentation of a science or technology from an historical point of view will be that which sheds light on process. Such documentation, of false starts, intermediate steps, ideas accidentally gathered during the course of other day to day activity, and personal recollections, is also the most fragile. Usually this kind of documentatation will survive in the context of undisturbed records but will not be retained after several moves, a change in ownership of the company, the retirement or death of the creator, or any conscious filtering by other than informed archivists. Therefore, the time to undertake such an effort is now, rather than later, and even at that, archivists will discover that much of the documentation is already lost.

The most important documentation from an evidentiary point of view

will be the record of software changes and change orders and the initial documentation of requirements and the programming specifications which were developed to satisfy those requirements. In effect there are always two evidentiary questions: did the software faithfully execute the intended procedures? and What information would normally have been available to a prerson using this system in the intended fashion at a given time?

B. SOFTWARE OPERABILITY

The barrier most often cited by archivists and museum curators to establishing a software archive or museum is the assumption that it will be necessary for the purposes of such an archive to preserve the software code itself in a form which makes it possible to run it on the hardware for which it was designed. If this was not such an insuperable stumbling block in itself, planners would soon discover that the same logic requires the preservation in operating order of all the software which ran in concert with the target package.

If it were necessary to retain some or all software in obsolete formats and to run it on the systems for which it was designed, the facilities required for any software archive would have to include a fully equipped, temperature controlled, fire protected, and heavily staffed computing center (of potentially vast size) and substantial expansion space for the variety of drives, CPU's , and I/O devices which were present in the systems environments for which the software was written. In this scenario the acquisition of applications will require acquisition of operating systems and language compilers which are synchronous with the application software release. Operating such systems would require all the usual expertise of a systems software staff, plus knowledge of a continuous stream of releases of each software component, for each manufacturer, if not each machine, in inventory. Obviously, deciding that it is necessary to preserve software in its context of operation in order to meaningfully study it is tantamount to determining that there cannot be comprehensive software archives for the full range of scholarly research topics.

I have concluded, however, that it is not necessary to be able to "run" the software. What, I began by asking, is the primary purpose of the archive? If the answer is public display or visitor education, then the software must run, or be emulated, to be appreciated. Since the answer is to support schol-

arly research or provide evidence of the functioning of an organization whose procedures are executed by software, the question is considerably more complex.

First we should consider just what is at stake. If we determine that meaningful kinds of historical research on software code require the preservation of associated software and hardware systems in working order, it is exceptionally unlikely that much software will ever be preserved, and that fraction which is will be retained, almost by definition, in museums. If, on the other hand, we save software code and documentation for which the enabling software and hardware tools are lacking, we must know what kinds of scholarly research can be conducted about software without running it and the kinds of associated documentary materials which will best support those types of research.

Let me define the issues, so as to make certain that we are not setting up a straw man. The debate is not about the value of programming code. I presume there is no debate over whether research can be conducted on the commercial distibution of software, or its legal protection, or the mechanisms of social recognition among communities of developers, or the impact of particular products or types of products on spheres of business. Surely no one will dispute that each of these kinds of research can be conducted without code, that each will contribute to understanding the history of software, and that, therefore, documentation to support these kinds of research are appropriately collected by a software archive. Many forms of material, from advertising copy to oral history tapes will contribute to this history, and the underlying story will be appreciated by laymen and will be accessible through exhibits, without recourse to the code.

One variety of scholarly research which will not be satisfied without recourse to the code is called "internalist" history of science or intellectual history. It is concerned with the development of concepts and the ways in which they are articulated. In order to know whether a software designer or programmer employed a paticular recursion technique or constructed independently executing modules, we need to examine the code. To understand a departure in AI programming, we may need to see how LISP demons were employed. However, the fact that many of the questions which intellectual historians might want to ask can only be answered from examining code, does not mean that they could be answered by running a program. Indeed,

the foregoing examples are illustrative of questions which would *not* be clarified by having the program operate; they can only be answered by dissection and analysis. One of the kinds of research which the software archive does want to support is precisely this type of intellectual history. It is for this reason that the historically oriented archive will often wish to acquire the code for specific routines, or modules, even though the systems of which they are components are not historically important or not available (the evidentiary archive would, in general, not retain such fragments).

At the same time, I will not dispute that even though the code, software documentation and advertising campaign records may be of value in understanding why one product succeeded where another failed, we will better appreciate consumer resistance if we could "feel" the product and see the way it worked. Nevertheless, I believe that published sources from the time and the correspondence of software company executives with their clients will point to the source of the market failure, even if we are deprived of a first hand "feel".

What, then, is the utility of saving the software (and the hardware on which it runs) in operating condition? What kinds of research can only be conducted, or can be conducted significantly easier, this way?

It is axiomatic among systems designers that one cannot fully understand a complex system by the study of its parts. Software is, of course, only one component in a complex system consisting of other software, firmware, hardware, communications media, standards, data, people etc. The kinds of interrelationships and dependencies which exist in complex systems are extremely difficult, if not impossible, to comprehend in the abstract. This is, of course, one of the reasons why software undergoes so many releases; capabilities which previously could not be realized become possible because of a change in the software or hardware in which the application is implemented. Changes to the application suggest needs to optimize features of the underlying environment which is then changed in response. As in any ecological system, these adjustments are taking place all the time.

The ecological metaphor suggests that researchers wishing to understand how systems actually worked would be seriously handicapped if the software could not be made to "run" as designed. But even retaining every release of an application system, and all the software and hardware releases which took place over its life, would not allow the researcher to model how

something actually worked at a specific site and time. Systems software specialists are notoriously clever at applying fixes to their local systems and equally poor at documenting them. The way to software actually ran at a particular place and time is nearly impossible to define, to say nothing of trying to replicate it. Even if we assumed that the "configuration management" history of a site was perfectly documented, it is extremely unlikely that we could ever reconstruct it, at least not without a large staff of systems programmers, and even then what we could conclude would be meaningful only for that site.

This is not to suggest that there is no value in seeing how a piece of software ran, or was intended to run. Much can be learned from such an experience, whether as a simulation, on film or by running it on the actual device for which it was designed. It is to argue however that too much can be made of the value of such "live" study of software for research purposes, and that with very few exceptions the costs of achieving such an end will be found to greatly outweigh the benefits.

If this conclusion is sound, there is little reason to retain software code in the original storage media or formats. If we don't have working seven inch magnetic drives attached ot the appropriate system with all the required software and output devices of the period, then we might as well take the seven inch tape and transfer its contents (while this is still possible using commercial service bureaus) to a medium which will be accessible to our research clientele. While code which is in print form need not be made machine readable, code which is already in a machine readable format should almost certainly be maintained in machine readable form on a contemporary medium, if possible (with the current preference being secured directories on hard drives or WORM drives). When the program is copied onto new media, and the original medium is an artifact of intrinsic historical interest, such as the Teletype Tape which input Bill Gates' BASIC interpreter for the Altair computer, the artifact can be accessioned into the Museum. In such a case both the copied code (used for historical research purposes) and the original artifact (used for exhibit purposes) would be kept. A similar case would be made for the first program commercially distributed on five and one-quarter inch floppies or some like distinction, but in most cases the original format is irrelevant.

Obversely, if the computers which ran a particular program are not in

existence anywhere, then retaining the system code would be futile if we believed that software had to be run to be studied. Those with whom I have discussed this issue seem to be in agreement that the argument for keeping code is actually stronger for such extinct hardware environments than it is for hardware environments which have been preserved or documented. While nearly everyone concurred that it would be acceptable for research purposes to retain only those modules, routines or even sub-routines which represented interesting departures with large scale systems written on computers which still exist or have been well documented, they saw benefits to retaining the complete code in the case of systems which don't exist and/or are poorly documented. The complete code will suggest the functions which lie beyond it; if these are running elsewhere, it is not worth keeping another copy, but if they are not documented, the code of an application which calls them may suggest aspects of their design.

C. THE LAW

Anyone considering creating a software archive or museum confronts a number of questions which can best be answered by an appreciation of the legal status of software as a protected asset. Can software owners and developers get protection for their products if they are deposited in an archive organized for historical research? What kinds of uses can the archive permit users to make of the software deposited with them? And who owns the software, and is therefore able to give it to the archive?

The same review of the legal environment surrounding software protection allows us to answer the question of whether copyright and patent office records will be an important source for identifying the universe of software and establishing what novel features of that universe should be considered for archival collection.

The authoritative source for any review of the legal issues surrounding software is The Computer Law Monitor (CLM). CLM reports on Federal court and state superior and supreme court rulings on matters of copyright, patent, trade secret and trademark protection for computer software and firmware (chips, microde etc.) as well as about many other legal matters relating to computing. Since laws, and court interpretations of them, can change, CLM should be considered a continuing source for answering the questions posed above.

As of mid-1987, certain conclusions are warranted. First, while copyright protection has been extended to all software programs in recent years (Apple v. Formula International), the situation has been sufficiently confused until 1985 to limit the utility of copyright office files for a comprehensive analysis of the software universe.

Secondly, the courts have recently upheld strict interpretations of the rights of copyright holders. They have ruled that software is tangible personal property (National Surety Corp. v. Allied Systems) and awarded significant damages for its misappropriation. They have made it clear that printing the code does not reduce rights to protect against its copying in electronic form (Micro-Sparc Inc. v. Amtype Corp., Apple v. Formula International), and they have ruled that reproducing the concepts and flow of code in another language (Whelan Associated v. Jaslow Dental Labs) or even making code from copyrighted English language statements of methods (Williams v. Arndt) is a violation of the act. They have further ruled that the programs need not have a copyright notice on them, if they are distributed with written material bearing the notice (Koontz v. Jaffarian). Finally, thay have applied copyright protection to video games (Midway Mfg. v. Dirkschneider) and other computer instructions so long as these are not determined to be the only way to produce a particular result. These protections should be adequate to assure the future protection of owners of software who deposit such materials as code and the records of its development in archives.

Thirdly, the courts in copyright and trade secret suits have provided positive incentives for archiving the records of the design and development process by ruling that in the absence of such records of independent development, charges that software was copied illegally can be upheld (Dickerman Associates v. Tiverton Bottled Gas). In other decisions about trade secrets, the courts have placed a substantial burden on the holders of trade secrets to demonstrate that they possessed a secret which gave them competitive advantage, that they took measures to prevent the disclosure, that they had a confident relationship with the party charged with disclosure and that when adopted for use the trade secret worked to their financial detriment. In the main, these requirements, along with growing protection in copyright and patents, may have reduced the appeal of trade secrets as a method of protecting software, but they also make it clear that where this method has

been employed, depositing the information in an archive, even with strict instructions to keep it closed until some future date, would risk the protection afforded by secrecy. In these cases it may be necessary for the archive to arrange for a future deposit, with the materials remaining in the custody of the owner, or a third party acting as the owners confident agent, until that date.

Fourthly, patent protection has been extended to ROM chips (Diamond v. Bradley) and may be extended to other forms of microcode (NEC Corp v. Intel Corp. still underway) despite earlier rulings that software, as a "calculation, mathematical formula or algorithm" was not subject to patent. Texas Instruments settled out of court with Fujitsu and Sharp early in January 1987 under terms which suggest that patent protections for RAM chips are now perceived to have real teeth. Now that computer code embodied in firmware is recognized as a "mechanism" and may be patented, we can expect the patent office files to become an increasingly good source for assessment of the direction of firmware development (especially since patentees must argue why their inventions are novel). Patent protection also depends on demonstrated development documentation, so the opening of patent protection also bodes well for archives.

However, the courts have not made the task of archives easier by their rulings on software ownership. In the case of Jostens Inc. v. National Computer Systems Inc. for example, they ruled that the purchaser of a proprietary software package owned the rights to the package as a whole, not to "the individual routines or lines of code", which, by default, belonged to the developer. In S&H Computer Systems v. SAS Institute, the court ruled that SAS was prohibited from copyrighting its software due to having accepted government funding early in the development process (before incorporating) which required products to be in the public domain. In numerous rulings on employment of technical staff, the courts have ruled that software products belong to employers, but that software concepts belong to their inventors. Since an historical archive will be interested both in collecting software products (a relatively straightforward ownership issue) and software concepts (a much more complex issue, since they must be embodied in code which may belong to someone other than the creator), they will need to be aware of these issues and seek the protection of joint donations when that is appropriate. The ownership of software developed under contract or by

employees of a firm may be clear, but the ownership of the copyright is not clear cut at all unless it was specifically given over to the firm which purchased the software or software license.

The upshot is that acquiring title to software may be a stumbling block for a museum or archival repository which wants to establish such a program. The developer, his or her employer and the client all have claims to ownership and each is also likely to possess different parts of the documentation. While such ownership rights are not legally any less straightforward than they are in any other kind of contracted creative production, these kinds of relationships, which have been exceptions in traditional archives, are likely to be very common in software collections. As a practical matter, this may make very little difference so long as the software being collected is obsolete. In such a case, employer and/or client rights are unlikely to be claimed when the source of the materials collected is the creator. Obversely, the creator would probably not retain rights in materials turned over to his or her employer or client, and would be unlikely to claim them against the archive in any case. However, anyone using the archive with the intention of publishing, would be well advised in any copyright situation, to seek all plausible permissions before reusing the material.

Even though the ownership issues are problematic it is not always, or even perhaps generally, advisable to wait until the commercial value of a software product has been passed to collect documentation. Much better records will be collected if the product is identified for acquisition early in its life. Indeed, it is likely to be impossible to find documentation of software once it has ceased being actively used[31]. In order to collect software related materials early in the product life without risk of liability the software archive will probably need to resort to some or all of the following tactics:

* acquire permissions from all parties which can be identified and located.

---

31 The author and Helen Samuels, archivist of M.I.T., recently visited the developers of the first time-sharing system, which was developed at MIT and used there for a decade. These individuals doubted that any documentation of that system had survived, and they had not substantially changed their methods even though they were now directing a project which has the self-conscious goal of defining the next generation of educational software.

* arrange for the acquisition of the materials at a future data when the party which has commercial interest decides that interest has elapsed

* acquire materials as they are being created but keep them closed to research for a period of time adequate to assure that the commercial interest has passed.

* make materials available, whenever they are opened, with strict warnings about ownership rigths in conjunction with initial registration of patrons, each specific request for materials from the collection, and any requests to duplicate portions of the materials.

Of course, the greatest protection derives from being granted ownership and all rights of use by those who previously enjoyed these rights. In acquiring holdings, the software archive should seek donations which grant the most comprehensive rights possible to the archive (but not necessarily to its patrons).[32]

Finally, the courts have been extremely clear about what constitutes fair use of protected (copyright) materials. In this they have clarified or a software archive what it may and may not permit of its users. Fair use includes any access which serves a public purpose such as criticism, comment, news reporting, teaching, scholarship, and research so long as the activity is not for profit and does not harm the potential market value of the work. Since any study of software and its development conducted in an archive would meet these tests, except for a use which resulted in copying the work for resale or reproduction in competing product (covered by the provisions of copyright) it would seem that a software archive is protected in the issue of user access simply by assuring that it is understood that all materials in its custody are covered by copyright - just as in any traditional archive. Exhibit uses, reproduction for explanatory, educational or scholarly quotation, and technical criticism would all be permitted.

---

[32] U.S. Congress, Office of Technology Assessment, Intellectual Property Rights in an Age of Electronics and Information, OTA-CIT-302 (Washington, DC, USGPO, 1986)

VII. POLICY FORMULATION

A. COLLECTION POLICY

As a consequence of forty years of rapid growth in the computer industry and in the associated scientific disciplines, there is substantially more information available regarding the history of software than any institution is in a position to review. Soliciting materials which cannot be properly reviewed or, if appraised cannot be stored or serviced, is professionally reckless. Therefore, the software archive needs to define explicitly what its collection policies are, and how it expects to fulfill its obligations to access the materials it solicits and to take the appropriate destruction or retention action with respect to them. It needs to implement these policies in procedures which can be carried out within the limits of its resources. And it needs to establish mechanisms to review and evaluate the policy and make changes when appropriate [33]

Because the collection policy is intended both to inform potential donors and professional colleagues, as well as to serve the staff as a yardstick by which to evaluate immediate acquisition decisions and longer term tactics and plans, it will be broad with respect to particular contents of the archives being collected, but explicit with respect to the purposes which the collection is to serve. The collection policy should address the geographical scope of the collection (worldwide, or U.S.?); the chronological scope (all recorded history, post 1940, post 1980?); the natural languages of the holdings (English only or all languages ?); and the media collected (printed, sound recording, film, photographs, machine-readable records- all?). It should explicitly identify any materials not collected.

The collection policies should address the relationship between the software archives and other activities of the repository, including its other collections, and its exhibits, publications, and outreach programs. While stating that the collection is open to all researchers, it should define the clientele which are anticipated to have the greatest interest in the collection. It should address cooperative collection development, resource sharing and information exchange policies. And it should define any external advisory

---

[33] Phillips, Faye; "Developing Collecting Policies for Manuscript Collections", American Archivist, 47(1), Winter 1984, p.30-42

structures which assist in setting collection policy.

In formulating such a statement, the repository should define a core collection, for which it maintains an on-going collection focus. For example, a dedicated software archive, such as that envisioned by the Computer Museum, might aim to collect U.S., post 1940, developments of non-proprietary operating systems and languages and defacto and de jure standards, and be continually seeking additional accessions to complete its documentation of these bases of modern computing. A software archive formed as part of a regional history collection might define its core collections as applications related software developed for in-house use by firms in its region.

It is important that historical repositories consider non-commercial, and even non-proprietary, software within their collection scope. Software systems and concepts which have become part of the public domain of computing, whether initially conceived in academia, in industry or in government, have been very influential in shaping subsequent software development. Non-proprietary software and the concepts embodied in it, are a community product, and have no other "natural" home, while the proper responsibility for documentation of the history of proprietary endeavors rests with corporate archives. Software systems which begin their lives as proprietary, may enter the mainstream of software discussion once they are successful, and the ideas they embody become public, not in the sense of ownership, but of discourse. Widely imitated operating environments, such as the IBM 370, are thus public in the sense that the ideas which are embodied in them are part of the background of every computer scientist, even if technically the operating system is still licensed and owned by IBM. Also, by documenting standards and widespread operating functions, software archives preserve a record of the fundamental structures of the software environment which will contribute to future understanding of more specialized software.

B. ACCESS POLICY

An organization dedicated to public education and the advancement of historical research has a clear obligation to make its holdings available to all on an equal basis[34] . This will have direct implications on the kinds of

---

[34] Smart, John; "The Professional Archivists' Responsibility as an Advocate of Public Research", Archivaria, 16, Summer 1983, p.,139-149

restrictions which it is acceptable to permit donors to place on materials and it is one of the reasons why a program of documentation of holdings is an essential component of an archive. The policy of the archive needs to be clear about when it departs from open collections. It should prohibit the acceptance of materials which will be closed indefinietly - specific dates and/or conditions should be attached to all restrictions. It should prohibit individual exceptions to collection restrictions and should make explicit the process whereby closed collections may be opened.

Implicit in what has been said about equality of access to the research facilities is the assumption that members, if there is such a category of users, would not have any special benefits in using the archives except that if the archives is located with the repository, and their membership entitled them to free admission or other benefits, this would apply to their use of member benefits on days when they were using the archives as well. Insofar as the archives charged incidental fees, these fees (reproduction charges, prices of publications, etc.) could be discounted to members without violating the spirit of open access. Policies which charge differential rates for essential services or access should be avoided.

C. COOPERATION

In exploring the concept of a software archive, we found that there were no organizations in the United States presently dedicated to creating such a research facility (with the possible exception of the Charles Babbage Institute whose mission is broader), and that general purpose archival repositories are either unaware of the need, paralyzed by the prospect, or committed to collecting machine-readable records in software independent formats. If the task of documenting the history of software is indeed larger than the capabilities of one institution, a substantial program of profession building and cooperative enterprises will be required to build the capability to adequately document the history of software. Organizations forming software archives need to adopt such an orientation as a matter of policy. The policy must explicitly be to encourage the preservation of materials of signifigance to the history of software, not just to collect them. Among the efforts it must support will be to publicize the reasons why collecting software is both desirable and possible, and to define the "universe of documentation" in terms of the issues and aspects of software history which consti-

tute a complete picture of the field. If the best way to encourage their preservation is through having others collect them (and this is frequently the case) then institutions need to try to adopt the longer term view and encourage the building of collections elsewhere.

A decade ago, the professional organizations most concerned about the documentation of science and technology in the United States (the History of Science Society, the Society for the History of Technology and the Society of American Archivists), formed a Joint Committee on Archives of Science and Technology (JCAST). One of the premises of the findings of JCAST was that archival materials have "natural homes", settings which contribute best to their interpretation, and that insofar as is possible, agencies should encourage documentation to be preserved in these "natural homes" rather than removing it from such contexts for the convenience of subject matter researchers [35]

Following the advice of JCAST requires that institutions cooperate in building collections. Cooperation does not end with cooperative collecting however; for to be successful it must spill over into the development of standards for description of materials, including, in this situation the development of authority languages for the description of software, and to the construction of information sharing mechanisms, newsletters and union lists, which can serve to alert the other members of the community of professionals concerned with software history to development in the field.

D. ACQUISITION POLICY

Acquisition policy governs a pro-active function, whereas collection policy passively demarcates a scope for acquisitions [36]. As such, an acquisition policy answers such questions as how a particular focus of collecting will be determined, what types of collecting tactics will be sanctioned, how emerging collection strengths are to be built, and what criteria are to be used in

---

[35] The position of JCAST, which reflects the authors views on archives of science and technology, is presented in: Elliott, Clark ed., Understanding Progress as Process: Documentation of the History of Post-War Science and Technology in the United States, Chicago, IL, Society of American Archivists, 1983.

[36] Anderson, R. Joseph; "Managing Change and Chance: Collecting Policies in Social History Archives", American Archivist, 48(3), Summer 1985, p.296-303

appraisal of accessions. As policy, it stops short of such procedural guidelines as how to approach potential donors and what kinds of agreements are within bounds.

The acquisition policy must address unsolicited offers and systematic collection building. It must establish how on-going collecting priorities are to be treated with respect to focussed research efforts or exhibit plans. And it must establish a framework for evaluating or appraising records which is cognizant not only of some "inherent" values of the material, but of their utility for the repository and its clientele and the likelihood that they will be used in this century [37]

First, an acquisition policy should identify the mechanisms sanctioned for active acquisitions efforts. For instance, if could identify the following:

a) solicitation of donations of materials which are related to the permanent collection interests of the repository

b) targetted acquisition by borrowing or purchase of documentation to be used in conjunction with an exhibit; and

c) surveying of materials associated with specific, funded initiatives.

Second, an acquisition policy must provide criteria for appraisal. For instance, it might establish that:

a) documentation is appraised with respect to its relevance to other holdings of the repository, its importance as a software industry standard, or its relevance to known research or planned exhibits. Such a statement severely restricts collecting, by providing explicit foci for the staff, but it leaves open virtually any arena for future collecting as the collection grows, research interests shift and exhibit plans reach out for new perspectives on the field.

A repository could decide to collect documentation regarding the development of software concepts and the impact of software products on society, but not collect documentation of software products in the form in which they were made commercially available or accession those records of software firms which reveal only the financial and managerial process. Such a policy both excludes the prospect of becoming a software library or a business history archive, and focuses the attention of the staff on intellectual

---

[37] Thompson, Gloria; "From Profile to Policy: A Minnesota Historical Society Study in Collections Management", Midwestern Archivist, 8(2), 1983, p.29-39

and social change agents, yet it is broad enough to permit collections incorporating some materials which will not be retained to be appraised.

Third, an acquisition policy needs to define a scope. For instance, it might include documentation of software which is used in any civilian contexts, or in civilian contexts other than financial or manufacturing industries except when software from those spheres illustrates a concept of important to an exhibit or developed in such away as to influence more general computing. Such a policy almost could exclude computing in government contexts, with similar qualifications to those above, on the grounds that public archives should take responsibility for documentation of governmental computing which is part of the public record [38]

The acquisition policy should also address scope with respect to the forms of material, physical formats, and volumes of records which are to be acquired because in accepting records the institution is making a commitment to provide access to them. As we have noted, this commitment can be a considerable obligation if the format is electronic. Providing the means to study the documentation of software will not always be easy, and collecting policy must reflect this consideration [39].

As an aspect of scope, the acquisition policy must address the extent to which the software archives will consist of published materials, including published software, and how such materials are to be acquired if they are collected (ie. if systematically, then their needs to be an adequate acquisition budget to support subscriptions and a mechanism for maintaining a reference library). If published documentation is not to be acquired except as part of an archival collection, the staff and researchers will still need access to a good library of reference tools and historical studies of software.

Fourth, it should address the openness of holdings, specifically whether any materials of a confidential nature, whether protected as corporate or government secrets or by confidentiality relationships should be acquired [40].

---

[38] Levine, David; "The Appraisal Policy of Ohio State Archives", American Archivist, 47(3), Summer 1984, p.291-293

[39] DeWhitt, Benjamin; "Long-Term Preservation of Data on Computer Magnetic Media", Part 1, CAN #29, Part 2, CAN #30, 1987. see also, Schenck, Thomas; "Magnetic Tape Care, Storage & Error Recovery", Library High Tech, 2(4), 1984, p.51-54

[40] Whyte, Doug; "The Acquisition of Lawyers' Private Papers", Archivaria, 18, Summer

Fifth, the acquisition policy must establish methods of acquisition; whether collections will be acquired only by gift, or if they will also be purchased, and if purchased, what kinds of materials would be considered for purchase and what proportion of the archive budget might be dedicated to purchase. The kinds of gifts which will be accepted, including bequests and donations of out-of-scope materials which might have commercial value, needs to be determined. The policy should address whether donations are acceptable under such special terms as requiring that all parts be kept, or that the collection be inventoried or exhibited. While the acquisition policy can always leave room for consideration of the exceptional case, it should provide sufficient guidance to staff on a day to day basis to permit them to act and to give potential donors an appreciation for the integrity of the organization and the objectivity of the appraisal process.

In this context, the policy should, finally, also address de-accessioning. Ideally, the policy would state that de-accessioning is an on-going function for the archive that holdings undergo regular review, and it would establish a mechanism for deciding what materials should be de-accessioned. Although it is a topic of hot dispute within the archival profession[41], there seems to be reason for a newly established collecting area, like software archives, to be more open about de-accessioning than others might be, since we have very little understanding yet about what kinds of materials will prove most important to research. It seems better to over acquire at this early stage, setting a time for review of the holdings which will permit us to acquire a greater perspective.

E. DOCUMENTATION POLICY

Some special practices will need to be adopted by a software archive to accommodate the requirements posed by cataloging and classifying the software documentation. These practices should, so far as possible, be built upon a base of standard archival practice, now best exemplified by the MARC-AMC format and associated rules for cataloging described by Steven

---

1984, p.142-53

41 Dowler, Larry; "Deaccessioning Collections: A New Perspective on a Continuing Controversy", in Nancy Pease, ed., Archival Choices (Boston, 1984).

Hensen and Elizabeth Betz.[42], These will be well suited to the needs of a software archive in all but two respects:

1) The software archive will need to determine how to catalog the software itself, insofar as it has software products among its holdings.

2) The software archive will need a classification scheme to index its holdings so as to support access by researchers with a variety of interests since the categories provided by the Library of Congress subject headings is hopelessly inadequate to discriminate between topics in this collection.

Conventions for the cataloging of software remain in dispute despite several years of library community discourse. Several cataloging traditions have met (head on) around the issue of how to catalog software. The community of data archivists, who developed the MARC-Machine Readable Data Files (MRDF) format in the late 1970's and the community of audio-visual librarians, who see software as simply another "special" audiovisual format are finding themselves badly outmaneuvered by the traditional book catalogers in the library community for whom commercially published software packages bear a great resemblance to books. A recent revision of the AACRII, Chapter 9, guidelines for cataloging software has unified the published item position but doesn't resolve the underlying issue. Staff of a software archive can follow this debate (hopefully to its resolution) in the pages of Cataloging & Classification Quarterly.

These library cataloging discusions are taking place independently from those in the publishing community, which is trying to develop a unique identifier for commercially published software equivalent to the International Standard Serrial Number (ISSN) or International Standard Book Number (ISBN). A National Information Standards Organization (NISO) Committee has been wrangling with this problem for several years and should report by the end of 1987.

Since the dedicated software archive will have no particular vested interest, its work could be greatly simplified by accepting whatever the

---

[42] Hensen, Steven; Archives, Personal Papers and Manuscripts: A Cataloging Manual for Archival Repositories, Historical Societies and Manuscript Libraries (Washington DC, Library of Congress, 1983).

Betz, Elizabeth W.; Graphic Materials: Rules for Describing Original Items and Historical Collections, (Washington DC, Library of Congress, 1982).

library community ultimately adopts, assuming (and this is supported by great historical weight) that the library community will immediately incorporate into its scheme any further data required or desired by publishers or vendors. This will, of course, not solve the problem for those software archives which are part of a larger organization and wish to see their holdings incorporated into institution-wide systems or databases.

The basis for a policy on indexing is even more difficult to resolve. Even though only Library of Congress subject terminology is recorded in the "subject" headings of bibliographic records, there are lots of other places indexing terminology unique to software collections could reside within library formats and library networks and perform its purposes for retrieval, if such a language existed. Unfortunately, no existing classification of software and software issues seems very useful. One reason for this is the rapid change which has been underway in software since its inception. No scheme developed to depict software at one time, or in one environment, seems very intuitive once we are removed from that moment or context. Since the absence of firm and widely held concepts is going to limit the transportability of any scheme, or its relevance over time, the repository should avoid fixed schemes.

Instead, terms from a number of facets of description should be assigned to each collection and related to a scheme which works at the present time. When that scheme requires updating, terminology embedded in the records can be left unchanged while external authority files are restructured to make the types of connections which seem best suited to contemporaries.

Three "perspectives" appear to represent software adequately and can guide us in assigning terms. The first perspective focusses on the documentation itself and employs form of material terminology to describe the documentation. The second perspective described the activity which the software supported and employs function terminology for authority control.

The third perspective focuses on the software which is the subject of the documentation and the activity and describes that it is "about." This perspective employs terms from the "latticework" for a software thesaurus sketched out in Appendix B. The schematic framework of that thesaurus is not fully elaborated, both because it is intended to be illustrative and because the use of the terminology from such a descriptive language leaves the typology open both to elaboration form within, by the addition of new facets

and terms within facets, and reconfiguration from above, when and if required. Staff of a software archive could use the "latticework" thesaurus, adding to it terms they for which they find "warrant" in the literature, to assure themselves that it can grow and be useful. Once a reasonable complete thesaurus has been constructed, the policy decision to adopt it can be made.

Note that the schematic representation is intended to only provide an hierarchical structure for the terms. Numerous terms may be assigned to the same item. The vocabulary does not locate any given collection or item in one place in the structure. The terms should be used rather than facet numbers, since the scheme itself can then be revised over time.

F. COLLECTIONS MANAGEMENT POLICY

Collections management policies govern the storage, retrieval, reproduction and preservation of the collections and types of records which will be maintained concerning actions taken in the management of collections.

Decisions need to be made in the context of the staffing and facilities of the software archive. Will documentation be stored in a separate facility? When and by whom will it be retrieved? What units of documentation will be retrieved (items, containers or entire collections)? What facilities will be made available for reproduction of archival materials and under what conditions? What facilities will be supported for the preservation of these materials and when will materials be considered for active conservation?[43]

Because the software archive will consist of both machine readable materials and more traditional paper, film and sound recordings, facilities will need to be provided for the storage of a variety of media. Acquisition decisions, already made, considered the most accessible media before deciding what to retain [44]. Retrieval of machine readable media at least (and other media if desired using optical scanning) will be from workstations attached in some fashion to data storage media on which the machine-read-

---

[43] National Archives and Records Service; Evaluating a Vital Records Program: A NARS Self-Inspection Guide for Federal Agencies (Washington DC, NARS, 1983) 14pp.

[44] Day, Deborah Cozort; "Appraisal Guidelines for Reprint Collections", American Archivist, 48(1), Winter 1985, p.56-63

able information is stored.

Original materials deposited or donated to the archive must be safeguarded. This requires a supervised reading area within the museum itself and a stack area which is staffed on a regular, if not constant, basis. Researchers frequently wish to make copies of materials consulted in libraries and archives for future reference. Normally this desire is accommodated by research facilities within the boundaries of copyright and on a cost reimbursable basis. The software archive needs to establish a policy governing the copying of materials (not otherwise restricted from being copied) and determine what, if any, basis is to be used to charge patrons for this service. The variety of formats of the projected archive create logistical problems which will need to be considered in developing procedures.

Most of the materials which will be received by the archives will have been written on paper. Until very recently, the paper stock used in computer facilities was as highly acidic as any ever made. Together with the extremely short half-life of information recorded on magnetic tape (stored under the best of conditions but not regularly rewound), the software archive can expect to inherit a large conservation and preservation problem. While these issues will be addressed further in development of procedures, the policy issue will need to be confronted at the time of accessioning (can the archives afford to accession materials which require conservation or will soon?).

Since the entire concept of collecting software is so new, it might be desirable for a software archive to establish a retention period for all accessions and re-appraise the holdings at that time. A reasonable retention period for material considered to be of archival significance today might be about twenty-five years, allowing the arhcivist to return to the material in 25 years with a fuller appreciation both of the evolution of the software industry since that time and of the holdings of the archive.

Another important aspect of a collections management policy relating to storage of the collection is to have a disaster recovery plan, based on the particular facitlity and staff and tested by at least a structured walk through on an annual basis.[45]

---

[45] Hendricks, Klaus B. & Brian Lesser; "Disaster Preparedness and Recovery: Photographic Materials", American Archivist, 46(1), Winter 1983, p.52-68

G. DISSEMINATION & LOAN POLICY

In the ideal future, software history will be well documented, and materials about most important developments will be cared for by widely distributed archives. Individual researchers will need to know what materials are available and how to obtain them, and will want to be able to view, borrow or acquire materials locally to avoid the expense of travel. Policies which will realize a system of inter-institutional cooperation and sharing of information will benefit researchers as will carefully crafted policies on remote access.

Archives in the United States have rarely established policies on the dissemination and loan of their holdings except to prohibit either. This is an unnecessary restriction, born of a confusion between the values which lead archives to accession documentation and the market concept of value. Since most archival materials retained for the informational or evidential value, that is for the data content, not for their artifactual or "intrinsic" value, there is no reason why they, or facsimiles of them, should not be distributed, with reasonable care. Because the users and staff of a software archive are likely to be able to exploit the potential of automation and communication technologies, the software archive is in an ideal position to experiment with facsimile transmission and on-line, even full-text, databases. Indeed, a sufficiently open policy on dissemination and loan of materials may be a route by which a software archive could avoid maintaining a reading room at all.

H. POLICY ON RESEARCH USE

Earlier, we observed that the distinction between a software archive and a software library was in part based on the purpose to which users put the software. In the software library, the software was used by patrons for the same purposes it was original sold to perform while in the archive, the software was studied to understand its origins or appreciate its effect on an application arena. In order to preserve its relations with vendors, and stand a good chance of receiving most of its software and related materials as donations, the software archive must rigorously enforce the use of its holdings for research purposes only. If the software is being used for the purposes for which it was intended (or is being copied by researchers and then pirated or used without a license) the owners will, with reason, be antagon-

istic to the archives and its purposes and might even resort to legal action. EDUCOM has developed a model policy for use by educational institutions, which could serve as a framework for a policy statement by a software archive, and the basis for an agreement with researchers.[46]

I. POLICY ON SPONSORSHIP

As long as corporate sponsorship for the archives is not tied to differential access to its holdings or special treatment for its records, there is no reason to establish a prohibition against any kind of corporate sponsorship or financial support. However, since the criteria for inclusion of materials within an historically oriented software archives are not the significance of the software product alone, and are definitely not an "endorsement" of the product, policies against permitting vendors to advertise that their products and associated documentation are deposited in the software archive would seem appropriate.

Finally, a structural feature of the marketplace for software might be employed by archives as a means to obtain some additional support from corporations. Software which is developed on contract, and licensed to users in a "custom" implementation, is often covered by a clause in the contract which requires the developer to deposit the source code in "escrow" in order tro protect the buyer in the event the company which developed the software should cease to do business. In addition, the nature of copyright law and patent protection is such that developerrs of software need to retain large bodies of documentation in order to uphold their rights to products. While somewhat unorthodox, it might be possible to mix these two functions while assuring the software archive of an early deposit of desired records. The software archive could agree to serve as escrow agent, keeping the source code closed except under the provisions of that agreement for a fixed period of time. In such an arrangement, the archive might also expect to receive payment for its services as the escrow agent.

---

[46] EDUCOM's statement on intellectual and property rights has been widely published and is beinbg incorporated into the policies of many academic institutions alongside their rules about plagiarism and falsification of data. For copies, write to: Steven W. Gilbert, Managing Director, EDUCOM Software Initiative, P.O.Box 364, Princeton, NJ. 08540.

VIII. PROCEDURES

Procedures are necessary (and should be required as a matter of policy), in order to assure that the activities of the staff of the software archive are conducted in a fashion which supports larger objectives. Procedures govern each aspect of the archives activity, but this study only addresses those areas where the procedures for the software archives differ from those of the museum or archival repository as a whole; for instance, it discusses qualifications of staff, but not selection procedures, issues relating to media storage, but not maintenance of the facility. Most of the procedures involved in administering a collection of software do not differ from those which would be in place in any library, archive or museum considering establishing such a collection. The correct procedures for any given institution with respect to software collections are those which are consonant with existing policies for other materials and collecting activities. A collection of general purpose forms which serve as samples for the administration of an archival program are available from the Society of American Archivists, as is a "basic manual" on the administration of museum archives and a manual on machine-readable records. Information readily available in these publications is not repeated here.[47]

Procedures are best when they are simple and logical. Unlike the typical in-house procedures manual, this report attempts to identify the rationale for its procedural recommendations. While this approach requires more discussion, it has the benefit of providing staff and management with a basis for adopting or not adopting individual recommendations, or for modifying them to fit the circumstances. Procedures should aim to assure that any information needed by the repository and its users is recorded the first time it is needed, and in a form which will make it useful and available for such future uses. Procedures should be tested under conditions which are as close to real as possible, before they are implemented. Tests should include time measures to be used as baselines for subsequent evaluation of the proce-

---

[47] Deiss, William; Museum Archives: An Introduction;
Hedstrom, Margaret L.; Archives & Manuscripts: Machine-Readable Records;
Archival Forms Manual.
All published by the SAA, 600 S. Federal St., Suite 504, Chicago, IL 60605

dures when they have been in place for a time and for improving the procedures at a later date.

A. PROCEDURES GOVERNING ACQUISTION

1. Solicitation

Any procedures for solicitation of materials need to assure that the process is targetted, efficient and produces no misunderstandings.

To assure that the targets are carefully thought out, some procedure should be established for a regular, administrative level, review of the subjects of proposed solicitations. Because collecting software documentation is likely to be only one component of the activity of the repository, decisions made at such a review about the subjects of all active solicitations should be made known to all members of the staff, especially including those not involved in the software archive itself, so that any opportunities for face to face meetings with the subjects, or interests of the subjects known to some staff members, can be taken advantage of.

To assure the efficiency of solicitations, a predefined format for solicitation letters, phone contacts and personal interviews should be developed and followed carefully. Because the terms under which software documentation will be acquired are, potentially, frought with legal consequences due to the uncertainties surrounding ownership and copyrights discussed earlier, departures from the form should be reviewed by senior staff.

To assure that there will be no misunderstanding, a single point of contact should deal with each potential donor once contact is established and the interests of the repository are explained. Terms of the gifts should be confirmed in writing before a gift agreement is presented for signature. The staff should visit the site where the materials are stored and access what is being offered both to plan for its shipment and to maie certain that materials being donated are indeed wanted for the collection. To protect itself legally and ethically, the archives staff should not ever provide or suggest a financial value of the materials being offered to the collection, although they might offer assistance in locating an appraiser. Shipping arrangements should be clearly understood by donors and checked in advance of the date of shipment. When magnetic media are included in a shipment, they must be identified for special handling.

2. Terms of Gifts

In general, deeds of gift should give the archives the right to do with the collection as its needs require. In the rare cases when special terms are necessary, they should be crafted to apply to everyone (except for legal heirs or the staff of an organization) equally. It is strongly recommended that the archive never accept any specific obligations, such as a description or exhibition schedule, with respect to a collection. Any terms requiring special treatment should be cleared with the director and/or the board, both for the protection of the staff, the archive and the donor. Arrangement requiring financial commitments beyond those normally incurred for a collection should always be made contingent upon funding. All departures from standard gift agreements should be filed so that future departures may incorporate the same terms when desired. This reduces the absolute amount of variation and could lead to the adoption of certain terms as part of a standard agreement, if, over time, there is call for it.

3. Accessioning Steps & Forms

Good archival practice dictates that the collections should be accessioned and described physically immediately upon arrival at the archive. Mass preservation activity, such as rewinding of tapes or deacidification of paper, might take place at this time, prior to description. Otherwise, more detailed description will be accompanied by appraisal and routine physical conservation at a later date. In general, materials of different media should not be dispersed until the collection can be described in full since they will contribute to each others appraisal. In collections of software documentation, especially, the existence of multiple formats of the same data may make decisions about what to keep easier; or the presence of materials of one medium may guide the archivist in retaining materials in another medium, as when having software makes keeping paper documentation essential.

B. PROCEDURES GOVERNING COLLECTION MANAGEMENT

1. Conservation

Practical conservation procedures for the software collection should emphasize proper up front condition documentation and conservation needs assessment. When the materials are received, the physical materials repre-

sented in the collection should be accurately identified and their present condition recorded. The hardware and software dependencies of all machine readable information storage media should be recorded in a way which permits regular, automated, review. The conservation program will then have available information required to determine whether the physical medium needs conservation attention or must be copied to another format either as a result of deterioration or of technical isolation. A "tickling" capability, to alert the staff of required, routine, maintenance, is essentially if any media are maintained in magnetic formats which will require regular attention.

2. Storage

Assuming that the archive takes the traditional approach of maintaining closed stacks, materials should be stored in random locations based upon space available at the time of accessioning. Since it is very likely that many media will be acquired in any given accession, and each will require separate storage conditions, we can expect that logical collections will be physically dispersed. Shelf location information in the online catalog should be restricted to authorized staff.

Open stacks arrangements are possible in archives, but rarely employed. The mixture of media in a software archive will make it extremely difficult to store materials in a logical order on the shelves or to provide the external markings necessary for patrons to browse or for staff to return materials to the shelves and spot misfiles.

3. Retrieval

Assuming the archive will adopt a closed stack environment, retrieval should be closed to staff not immediately responsible for such duties as well as to researchers. This protects the staff from any responsibility for materials which disappear and makes the maintenance of statistics and stack conditions the responsibility of one unit or person. The requirement to retrieve materials in order to consult them for internal reference purposes as well as for outside researchers should be conducted using procedures which will also capture information relevant to evaluation of all aspects of the program. "Call slips" or their equivalent, should be designed to capture not only the dates of the retrieval and what was retrieved, but also information about patrons and their uses of the materials, as well as additional, user suggested indexing terminology.

While the decision will necessarily be made based on available local resources, it will often make sense for a software archive to copy machine readable data onto optical digital disks. Some consideration should be given to the possiblity of scanning all archival materials into this format depending upon whether favorable terms can be agreed to for a scanner, thereby reducing retrieval requirements for machine-readable data to an absolute minimum.

C. PROCEDURES GOVERNING ACCESS

1. Reference Procedures

Reference procedures for the software archive need not differ from those of other archival programs. However, the archive should determine its policies on the distribution of single copies of software and implement these in its procedures.

2. Finding Tools

Cataloging procedures will need to be established which satisfy the need to a software archive to have integrated access to all of its holdings without respect to the form of material being described. The only acceptable framework for such catloging is the MARC-AMC format rather than the specialized formats for machine readable data, audio-visuals, and published texts. If possible, reports on holdings should also be contributed to a national bibliographic network (OCLC, RLIN or WLN). A case can be made for a specialized national union list for software and related documentation, similar in some respects to the National Union Catalog of Microfilm Masters or the National Union Catalog of Manuscript Collections.

3. Reading room use

Reading room procedures for the software archive do not differ from those of other archives except that if PC's or printers are provided or permitted, users should be prevented from copying software code. Disc duplication and print output directed to the reference desk will probably serve as a sufficient control.

4. Loans

Any given record from the software archive, as in most archives, is likely to be in low demand. The software archive can, therefore, consider proce-

dures for the loan of collections (and parts of collections) to other repositories where the materials can be used under the supervision of another archivist. A procedure which recovered the costs of making a digital image of materials to send would have the benefit of making future loans of the same materials easier.

5. Copying Procedures

Legal restrictions on software copying are no more stringent than that on other forms of copyright, but the archive should explicitly inform users that possession of a legal copy made under the research fair use provisions does not give them the right to make additional copies or permit them to use copies which have been legally made for research, for the software's original, intended, application. Users will not generally understand or appreciate these restrictions on software even if they are familiar with copyright restrictions on other materials.

D. STAFF

1. Regular Staff

Of the various staff positions which might be important to a software collection program, only that of the curator/archivist of software needs to be specific to this function.

The position should require:

* An advanced degree in library or information science, the history of science or technology, or museology
* Experience in archives, archival automation and management of machine-readable records

Selective factors would include:

* Knowledge of museums and the role of archives in museums
* Appreciation of the nature of disciplinary history centers
* Knowledge of the nature and history of software
* Experience in managing multi-organizational networks.

In order to attract reasonable candidates, it would be useful to post announcements of the availability of the job both in the Newsletter of the Society of American Archivists and SPECTRA, the Newsletter of the Museum Computer Network

2. Affiliates

A program which encourages serious researchers to spend substantial time at the archive conducting their research is of benefit both to the archives and scholars. Procedures for "affiliated" researchers, whether on sabbatical from large computer firms which have such leaves or on fellowship from universities, should be established. Such affiliations need not carry any responsibilities or rewards, but should be reviewed by a committee imbued with some prestige.

APPENDIX A

Position Description

Curator/Archivist of a Software Archive

Introduction:

The curator/archivist is responsible for developing and carrying out the program of the software archives. The incumbent will represent the institution with potential donors and the professional community as well as to researchers. The incumbent will establish and maintain policies and procedures, within the framework of the administrative practices of the rest of the organization, which are consonant with the needs of a software collection, and which achieve its aims in a professional and accountable fashion.

Duties and Responsibilities:

1. Sets program objectives and allocates resources:
   * Drafts plans and budgets with appropriate advice from colleagues inside and outside the organization, for submission to management.
   * Assures that resources are assigned to carry out approved projects.
2. Builds Collections:
   * Articulates annual collection development objectives within the scope of the collecting policy.
   * Assists targets of collecting objectives to deposit records, soliciting their donations of materials to the software archive as appropriate.
3. Manages Collections:
   * Accessions, describes and provides access to holdings.
   * Publishes descriptions of archives and assists in selecting materials for exhibits.
4. Develops Professional Awareness of Software Archive Issues:
   * Gives talks and publishes in the professional literatrure.
   * Participates in standards committees for software cataloging.
   * Participates in efforts to forge inter-institutional exchanges of software archives holding information.
5. Interprets the History and Sociology of Software:
   * Assists in developing exhibits and public programs using the materials of the software archive.

Supervisory Controls, Guidelines and Contacts

The incumbent will operate as a member of the senior professional staff of the repository, with general supervision and periodic review, but largely following the adopted plans and broad professional guidelines. The specific guidelines for operating a software archive contained in this report serve as the only direct source of advice, and are subject to substantial evolution and professional consensus. The incumbent will be expected to play a role in the further development of professionally accepted guidelines for software archives and exercise good judgement in the interim.

The incumbent will need to contact a wide range of individfuals and organizations on behalf of the repository. These contacts would include, but not be limited to, vendors, potential donors, researchers, colleagues and staff. International contacts would not be unusual and inter-institutional agreements would be a frequent topic of discussion with contacts.

Qualifications:

In addition to an advanced degree in archives or library administration and/or computer science or equivalent academic qualifications and the specific prior experience implied above, the incumbent must possess:

* Broad knowlege of the history of software and the nature of computing languages, adequate in depth to understand software and its significance, but not necessarily to program.
* Demonstrated ability to develop new programs from general guidelines.
* Experience in establishing and administering archival control systems.

APPENDIX B

Latticework for a Software Thesaurus

This framework for a language to describe software from a variety of different perspectives is being proposed as a vocabulary for use by a software archive, but rather as a preliminary model for building such a vocabulary. Terms in this example were found in current literature regarding software, that is they all have warrant, but numerous terms which could also be included were not. Lacking the facilities to maintain a comprehensive vocabulary for software, the author felt that it would be better to place before his readers a structure which was clearly incomplete, rather than risk misunderstanding by presenting a more complete vocabulary which was then taken as definitive. The point of this framework is that it can be built upon; new major facets may be added (A-Z), including any referenced in the section XX of the report, and new terms within categories will need to be added, including those for lists (such as languages, or operating systems) which are clearly incomplete.

A. Academic Disciplines
- *Cognitive Science
- *Computer Science
- *Information Science
- *Robotics
- *Software Engineering
  - **Development Methodologies
    - **Software Architectures

B. Software Business
- *Software Advertising
- *Software Authoring
- *Software Marketting
  - **Software Licensing
    - ***Shrinkwrap Licensing
    - ***Site Licensing
    - ***Run Time Licensing
  - **Software Sales
- *Software Purchasing
  - **Requests-For-Proposal
  - **Software Competitions

*Software Trade
**Software Trade Balances
**Software Trade Policy

C-K available to be added

L. Software Literature
*Technical Literature
**Primary Literature
***Software Books
****Text Books
****Technical Books
***Software Conference Proceedings
***Periodicals
***Software Technical Reports
**Secondary Literature
***Software Directories
****Printed Software Directories
****Online Software Directories
***Software Review
**Tertiary Literature
***Software A&I Services
*Lay Literature
*Software Humor

M-N available to be added

P. Software Political Context
*Software by legal context
**Software Legislation
**Software Regulation
**Software Registration
***Software Copyright
***Software Patenting
***Software Trademarks
*Software Industry by political context
**Software Lobbying
**Software Liability

Q-R available to be added

S. Software (itself)
*Software by application (or function)
**Data Management Tools
***Database Management Systems
****DBMS by Type
*****Hierarchical DBMS
*****Networked DBMS
*****Relational DBMS
***Data Dictionary and Data Models
**Data Processing Software
***System Management Software
***System Acceptance Tools
****Benchmarking
****Data Comparitors
****Reliability Testing
***Configuration Management
***Software Development
****Debugging Software
****Design Software
****Documentation Software
****Prototyping Tools
**Entertainment Software
***Games
***Hobbyist/Leisure Software
***Interactive, Multi-media Software
**Integration Software
***Software conversion tools
***Peripheral Drivers
**Monitoring & Instrumentation Control Software
***Environmental Monitoring Software
****Energy Management Systems
****Security Monitoring Systems
***Machine Control Software
**Office Automation Software
***Integrated Office Systems
***Word Processing Software
***Scheduling & Calendaring Software
***Voice Mail Software
**Graphics Software
***Business Graphics Software
***CAD/CAM Software
***Animation & Simulation Software
***Graphic Image Manipulation

*Software by collectivities or components
**Software Code Collectivities
***Libraries
***Routines
***Modules
***Sub-routines
**Software code functions
***Addressing Schemes
***Interfaces/Calls
**Software Code Types
***Algorithms
***Formulae
***Protocols
*Software by computing role
**Operating systems
***Operating systems by name
****360
****VMS
****Unix
***Operating subsystems by function
****Communications Management
*****Time Sharing Systems
*****LAN Operating Systems
****Memory Management
****DASD Management
****Input/Output System Control
**Compiliers by Language
*Software by context of origin
**Industrial Software
**Military Software
**Office Software
**Personal Software
*Software by Dependency
**OS Dependency (by name)
**Language Dependency (by name)
***Ada
***ALGOL
***Assembler
***Basic
***C
***COBOL
**Hardware dependency, by component
***Mainframe Software

```
                ****IBM Processors
                    *****Model 1401
                ****Control Data Processors
            ***Mini-computer Software
            ***Micro-computer Software
                ****Multi-user Microcomputer systems
                ****Single user microcomputer systems
            ***Data Communications Equipment Software
                ****Local DCE Software
                ****Remote DCE Software
            **Standard Dependency, by type
            ***Communication Standard Dependency
                ****TCP/IP Dependent
                ****SNA Dependent
    *Software by Developmental Stage
        **Released Software
        **Pre-release test software
          ***Alpha Test phase
          ***Beta Test phase
        **Vaporware
    *Software by Format
        **Source Code
        **Object Code
        **Microcode
        **Firmware
    *Software by Information Management Strategy
        **Artificial Intelligence Software
          ***Expert Systems
          ***Robotics
          ***Lingistic Analysis Systems
        **Information Retrieval Software
          ***Full-Text Retrieval Software
          ***ISAM Retrieval Software

T. Software Training & Teaching
    *Software Training & Teaching Programs
        **Degree Granting Programs
    *Software Training & Teaching Materials
        **Software tutorials
        **Software courseware

U-Z available to be added
```