THE CONCEPT OF A SOFTWARE ARCHIVE:

A REPORT TO THE COMPUTER MUSEUM

PLANNING FOR A SOFTWARE ARCHIVE - PHASE 1

ARCHIVES & MUSEUM INFORMATICS

JANUARY, 1987

THE CONCEPT OF A SOFTWARE ARCHIVE
TABLE OF CONTENTS:

Executive Summary

In 1986 the Computer Museum contracted a study of the feasibility of establishing a software archive as the first phase of implementing such a program. Because the cultural and historical significance of forty years of computer programming was self-evident, and no library, museum or archive known to the staff of the Computer Museum was collecting software for the purposes of historical documentation, the questions of whether it was possible to create such an archive, what it would look like and whom it would serve, needed to be addressed before planning could commence.

Two potentially insurmountable technical barriers had been previously identified which the study needed to address: 1) was it necessary to be able to run software in order to serve an archival purpose and 2) were there substantial legal barriers to receiving software archives or making them available for research. These two questions, addressed in Appendixes 1 & 2, and are both answered in the negative. Important historical questions can be answered from a range of software documentation without recreating the environments in which the software ran, and recent tightening of copyright and patent protection has significantly improved the likelihood that owners of software would consider depositing it in archives and that archives could, ultimately, make such materials available.

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

- what is the domain of a software archive?
- how can an institution frame an executable collecting policy (for its own program and with other repositories)?
- what is the mission of such a program?
- who are its users and what are its uses?
- what steps need to be taken to establish a software archive?

In a synopsis of the history and sociology of software development and distribution, numerous historical perspectives are identified. The potential domain is found to be well beyond the scope of any individual repository, so criteria for collecting foci are proposed and cooperative reporting suggested. Varieties of documentation are reviewed and their value for particular types of historical queries suggested. The value of such a program is identified as being primarily the support of historical scholarly inquiry, both from the nature of its potential users and their uses and from its limited value as exhibition material and for public education.

The report concludes that such a program is possible and would be advantageous, and that the Computer Museum should proceed with its further definition in two planned studies which address the policies of such a program and its operating procedures.

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 to 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 of our day, yet their content, structure, cultural impact and socio-economic significance have not yet become the subject of critical-historical study. Indeed, early examples of this genre, monumental intellectual achievements of early software development, 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, determined to establish a software archive, if it was feasible. They contracted with the author to examine the nature of software and its documentation, and determine the feasibility of such a program. The study is to be conducted in three phases, the first of which is a proof of concept. The second and third phases, if called for, will be a plan and procedures for implementing and maintaining a software archive. This document is the report on phase one - the proof of concept. It defines what a software archive might be and endeavors to demonstrate that such a program is not impossible and would have some substantial cultural value.

This report poses broad questions about software, about documentation efforts and about the requirements of such a program, and resolves these questions when it can be shown that they are not fundamental barriers to proceeding. It does not lay out costs and benefits or identify the guidelines which would be used in administering such a program. These would have to be addressed by any given institution in determining the scope, if any, of its own software archiving program, but they are more suitably addressed in a planning phase report, and will be if the decision is made to proceed.

I. The Domain of Software: What could be archived?:

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.

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 imbedded in hardware, reflects the increasing resolution of this still gray area. For the purposes of a survey of the universe of potential software documentation issues, software must be considered to include any instruction set, however emboddied, thereby forcing us to consider (although not necessarily to 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 after which computer programs can be considered software; the potential domain of a software archive must extend to the earliest computing devices. Experimental computers 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 EDVAC in 1951), and paging of memory (although nmot 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 very important software concepts emerged from this period. In documenting the history of software we cannot ignore these developments simply because the term "software" had not been coined, or because executing a particular program on these machines required a particular 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-standard, and tied closely to the physical device, these programs and their 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 up, 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 the 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 largely confined to the province of the computer manufacturers, in collaboration with their (large) customers, for several years. 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] 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 for this period 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 the size of early custom systems and their huge costs have helped to assure a documentary record. Preliminary inquires failed to find documentation 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 disk packs (introduced by IBM as RAMAC in 1956), the computer became a sufficiently general tool to require local engineers who were capable of writing instructions 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. Such applications in themselves, together with the development of general languages for writing machine independent instructions, 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), 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 the reflection of a manufacturers requirement to install 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, each language envisioned as being most appropriate to particular kinds of problems and each providing tools of particular sorts. Thus the history of programming languages is particularly important at this time; fortunately programming languages are the one area in the history of computer programming for which a modest archive has already been established.[9] The emergence of programming 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. The developing public awareness of software, and of the ways in which software could be used to competitive advantage, is a critical component of 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 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 of instructions required had grown from a modest 5,000 lines for the IBM 650 to several million lines for the IBM 360![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 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 contrast 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 mechanisms 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 from them. In these functions the computer performed work which could have been given to people, but was repetitive and dull. Routine bookkeeeping on a large scale was error prone; searching for files of cases was likely to result in misfiling them after the tasks was 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 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 were already 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, and altering the balance of power between information rich and information poor.[13]

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, and they were numerous and thus

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 competitive 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 books 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 numerous national security 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 application require entirely new computers to be designed especially for them; thus the market continues to 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 implementation 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 with increasingly specialized roles assigned to participants and growing degrees of similiarity 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 identifying software efforts during their 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 challenges 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 governing of storage heirarchies and input/output devices required increasingly complex operating systems. Over time, some specialized applications could be better performed with machines designed to satisfy applications requirements at the hardware level without the overhead of general operating systems software not used by the particular 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, not how it should use the materials or who it should serve. The role of historical understanding here is to sketch a landscape; it permits rational choices to be made in the upcoming journey. Like any sketch, it also does not identify the specific foci of collecting which will best illuminate any particular part of the picture, but poses a challenge to the archivist to begin to document more fully, and learn more detail which then guides the next stage of collecting.

II. Collecting Policy:

A. Perspectives on the history of software:

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 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 collecting policy begins with the definition of a schema which adequately depicts the universe of which the collection is to be a subset.

Traditionally, software has been classified somewhat unidimensionally, and from the perspective of the machine, according to the layer at which it operates with respect to other software - ie. 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 a more comprehensive schema, such as that used by the ACM Computing Reviews? No. 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.[15] 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, 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?

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?

b) Communication Environment - how is it accessed? (direct connect, timesharing, remote terminal, LAN, WAN, VAN? etc.

c) Software Environment - what virtual machines? Operating Systems? DBMS and I/R systems, development tools etc?

The problems we will encounter here are the exponential increase of configuration options if total systems environments are targetted.

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 broadbased 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?

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

11. Social impact - What populations or events were effected? The challenge here is to examine impacts not just on pulic 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 it 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 by ownership restriction materials which might not be collected as documentation of some other view.

B. Forms of Material:

Up to this point we have spoken of software documentation as if it was uniform and identifiable. It is not. 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 consist 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 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 criteria and a 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 attendence - 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. While such a schema will need to be developed fully, if we procede to establish an archive; an unsystematic discussion of the kinds of points it would address is presented in Appendix 4.

One additional point should be made about documentation, because it poses a serious programmatic choice for any software archive: if the documentation does not exist or was never created, reconstructive research, including oral history, model construction 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 archives choose to support active documentary reconstruction efforts. Whether this form, of material will be collercted must be addressed directly.

C. Selection Criteria:

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 will face 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 is not very profound, but might easily be missed.

Nor is the first criteria 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 types of networks which participated in the elaboration of the clever routine, we may be able

to document a considerable amount without the code.

Selecting criteria are not, however, usually so obvious to derive or simple to apply. Should a repository seek out the earliest manifestation its focus? Or its most successful embodiments? Or manifestations which are internally interesting, coherent or rich? Should it collect an instance from 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 local 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 practical 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 the framework of other existing documentation. It 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 sources, published and unpublished.

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 from 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 consequence, documentation of a software product over a period of time can expose different factors 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 other 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.

Criteria for collecting are not dictated by abstract values, but reflect the actual interests, expertise, limitations and strengths of the collecting repository. 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 for the Computer Museum will be to define collecting policies and selection criteria which fit smoothly into the overall goals of the institution.

III. Extant Documentation & Repositories of Software: Where is software?

In undertaking to establish a software archive the Computer Museum was acting on its belief that such collections do not already exist and that there remains extant documentation which deserves archival retention. The existence of other repositories interested in or already pursuing software archiving would be welcome; the size of the universe has already determined that no one archive can be comprehensive. Nevertheless it is ctitical for the development of rational collecting strategies to know of other repositories interested in, or collecting, the history of software. Likewise it is important to have some sense of the extent of documentation which might be discovered by a full blown program to archive software and its documentation. In addition to the literature search which was conducted at the outset of this study to identify works on the history and sociology of software, I conducted a phone survey of archivists at a few computer companies, high-technology firms, universities, professional associations and governments to determine whether they collected such material, and whether that thought the records of software history still existed in the institutions for which they were responsible. In addition, I tried to determine why they currently were not collecting this material, in order to identify any previously unforeseen obstacles. (The contacts and their institutions are listed in Appendix 3.)

Though superficial, this survey exposed some salient facts about the enterprise:

1) As assumed, no one has formed a software archive nor is anyone collecting software documentation except as an inadvertent bi-product of institutional archives. Even organizations such as SHARE and the Federal Software Exchange, which existed for the purposes 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, Waterloo, SHARE, FSE[16] etc.) would probably be very conducive to depositing their materials.

2) In the few repositories which are sophisticated in collecting machine readable materials, attention has been exclusively focussed 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 has been in 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. Here also, 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.

3) In general the concept of collecting software for historical research purposes has not occured to archivists, in part because there is no user community. It seems no one ever asks for such documentation! For a variety of reasons best traced to the influence of 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.

4) 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. Initially each archivist felt that executing the software would be necessary, and

couldn't imagine how to provide such facilities. 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 example, however.

5) Once introduced to the concept, 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 consortuium or reporting network.

While little was revealed by the interviews which was not anticipated, the absence of a community of scholars does need 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 would 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. Archivists intuitively regard a software archive as consisting of code, and are therefore likely to discard other

documentation even if they are trying to be helpful. Since we are speaking of archivists here, we can't expect that sending the AFIPS brochure "Preserving Computer Related Source Material" will be the solution. Getting archivists to preserve appropriate source materials depends on an appropriate articulation of the mission of the software archive. It will also require planting the concept of the history of software as a broadbased resdearch arena in appropriate places in archival literature. For example, Appraising the Records of Modern Science and Technology: A Guide, though only two years old and written by an active participant in the software archive movement, barely refers to computer software and doesn't suggest the value of documenting it as subject rather than object of the scientific endeavor.[17] 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.

IV. The Mission of a software archive or museum:

A. Distinctions between a software archive and a software library:

The first step in establishing a software archive, 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 archive or museum differs from a software library in that its primary aim is to document the history of software, while that of the software library (even if, following data propcessing usage, it calls itself an "archive") is to provide software for the shortterm use of its clientele. As such the archive supports research about software, not research using software.

A software archive is an historical collection. It may document how particular institutions used software, and thereby provide evidence of the functioning of the organization, but this is a secondary purposes of the software archive per se. Of course, evidential reasons may be the primary or even sole reason why a general archival program retains a particular software product. (Indeed, I would argue strongly that corporate archives should retain software documentation for evidential reasons, but this report is not about such activitites.)

Because its focus is on exemplars of a creative genre, rather than on evidence of the activity of an organization or person, the software archive resembles a literary, art or music collection. It collects specific examples of software and materials which provide

evidence of the way in which software was created, distributed and used, in order to support research on software itself. However, 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 records of the software products alone, without any documentation of process.

Even though a software archive or museum will seek to further our understanding of the history of software through interpretative exhibitions and publication, its principal function is to preserve a body of material for research. Although exhibition purposes are best served when we can show as well as tell, a researcher will be able to learn much from 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 without necessarily playing it back on the machine for which is was written. Therefore, 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. [For further discussion of this issue, see Appendix 1]

In this respect, 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. Further, the archive or museum is interested as often as not in only 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 for the kind of small, general purpose, computers it can afford to maintain for its users.

Finally, 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.

B. Acquiring the software archive:

Except for those organizations in the software business, the acquisition of a software archive will require "collecting" rather than "retaining" materials. This discussion is restricted to software collecting, since the purposes of retaining software, as evidence of organizational activity, are distinct from those which guide collecting.

There are several discrete acquisition issues:

1) what do we want to collect?

2) where is it found?

3) who owns it?

4) how may it be acquired?

Previous sections of this report addressed the first two questions. We want to document the history of software, not an unproblemmatic goal, but given the magnitude of the effort, one which no single archive, or small group of institutions, could achieve without cooperation. Efforts at pre-arranging what independent institutions will collect have been notorious failures, so no effort will be made to "divy up" the universe. Instead, the 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 . In this way they can controls their own collections strategies while cooperatively covering the field.

The second issues is where we will find the documentation. In the first section we hinted broadly that it would be found along the dissemination paths. Just as the most interesting documentation for

understanding the process of creation of software will be found closest to its origins, documentation of other significant processes (distribution, sale, use) will be found closest to where they took place. 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 education, or air and spaceR&D)[18] some towards internal developments in software concepts. Any comprehensive long term documentation strategy will necessarily require numerous organizations with such distinctive perspectives to cooperate.

Acquiring title to software may be a stumbling block for a museum or archival repository which wants to establish such a program. [For a detailed discussion of legal issues surrounding rights in software, see Appendix 2.] The developer, his or her employer and the client all have claims to ownership and is also likely to each 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 have traditionally been exceptions in archives but they 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, the employer and or client rights are unlikely to be claimed if the source of the materials collected is the creator. 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 as in any copyright situation, to seek all plausible permissions before reusing the material. Nevertheless, the ownership issues are likely to be problemattic because it is not always, or even perhaps generally, advisable to wait until the commercial value of a software product has passed to collect documentation. Much better records will be collected if the product is identified for acquisition early in its life.

In order to collect software related materials early in the product life without risk of liability the software archive should probably resort to all of the following tactics:

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

2) arrange for the acquisition of the materials at a future date when the party which has a commercial interest decides that interest has elapsed or acquire the materials but keep them closed to research for a period of time adequate to assure that the commercial interest has passed.

3) make materials available, whenever they are opened, with strict warnings about ownership rights 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).[19]

V. Users and Uses:

An archive for the history of software could attract three kinds of users:

1) Regardless of how it 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 collection 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) 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 reusable code is still a matter of debate within the software community, the potential value of studying previoyus 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) In theory, the software archive could be a facility for casual, 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 this kind of casual inquiry is 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 domain 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 which generated them, and in the acquisition of movies showing how software operated on machines for which it was initially designed. Another exception is in timed demonstration disk and educational software ventures, both of which are becoming a more common, but still represent only a tiny corner of the market.

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 to the Museum. As noted earlier, 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 such collections.

One component of the software archive program, then, ought to be directed at 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 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 enmploying graduate students to encourage dissertations using software archives resources. Possibly 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 the Computer Museum. and other 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 Frameworks and Timetables:

How then, should a particular institution, in this case the Computer Museum, 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. [In a different situation, in which the software archive was to be an independent entity, it would need to adopt a statement of goals and develop collections policies.] The critical factor in this decision is the recognition that a software archive is a commitment to supporting research; while it mnight contribute somewhat to exhibition and public programs, the principal benefits will be long-term and academic, and the burden does have the potential of taking resources away from other museum programs (although, as discussed later, I believe a software archive can be self-supporting).

While the Museum can announce its openness to receiving software and software documentation, it should not do so without a clear sense of the criteria it will use to evaluate such gifts. In effect, these criteria are identical to those it would employ in a self-conscious collecting effort. Such self-conscious collecting efforts will, in any event, be required in order for the Museum to acquire a collection with coherence and roundedness. 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. To develop such criteria, the Museum should articulate what it expects its archives to be and to be doing in ten years.

Defining the significant developments and actors in any given plane of the matrix of perspectives on the history of software described in section 1, is a substantial research effort. While the Museum is well positioned to undertake such research, and it curatorial staff needs to do so for general purpose computers and general purpose software as a consequence of its mission, it should use specific exhibitions, or funded research projects, or special publications to frame software collecting objectives in specific arenas. Initially the program was envisioned with a half-time curator; I believe that this will not be adequate to give the program the level of activity required. A full time curator in charge of research and solicitation, a part-time collections processor and significant clerical support, would seem to be a minimum for permanent staffing. Consideration should be given to strategies which would augment such a basic staff with additional focussed resources.

The rhythm of such special projects in an organization the size of the Computer Museum can be estimated from the activity of the very successful Center for the History of Physics at the American Institute of Physics in New York. The Center, founded in 1965 at the end of a four year "survey" project 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, phsdics in national laboratories, and laser physics - with support from outside 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 criteria for assessing 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 responsible 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 scientists have proved very supportive and the Center has also made some fundamental contributions to archival practice.

A reasonable long-term strategy for the Computer Museum could be developed around this model. During an initial period of three years, the Museum could form an archive devoted to documenting critical developments in general purpose computing since the earliest computers, but focussing active collecting on the past twenty years.

It would agressively collect information about holdings already deposited elsewhere (if any can be found) and accept general purpose software of an earlier period, if offered. After three years the Museum could launch a separately funded project to document the early evolution of software (prior to general purpose mini-computers and the IBM 360), with special funding from major foundations and the industry. This effort would have publication of an early history, an exhibit, and the articulation of explicit collecting objectives for future pre-1960 collecting by the Museum, as its products. Subsequent special projects might include a focus on a particular industry (aerospace, finance, insurance) or a sector of the economy (government, corportate, non-profit) or a specialized software function (CAI/CBE, CAD/CAM, AI, Graphics). Again, these would be coordinated with exhibits and publications, and would have explicit goals of alerting potential future donors to the interests of the Museum as well as targetting specific items for acquisition.

If the decision is made to go ahead with a software archive, the Museum should launch a concurrent campaign to raise additional funds for the core program. While the reasons to collect software are intrinsic, collecting it could open new sources of support for the museum. Only a handful of software companies are represented among the contributors to the Museum at present, even though the universe of such companies is much larger than that of hardware firms which are well represented. While the engineering of new computer systems is an anonymous undertaking, the authors of software are still frequently sole operators or leaders of relatively small design teams, and take a considerable pride in the products of their

efforts. Both the software companies and the authors of software, need to be reached in order for the software collecting effort to be successful; it would be very surprising if these contacts did not generate substantial new gifts of funds in addition to the software documentation targetted for collection acquisition.

Generally, the most interesting documentation of software development from an historical point of view will be that which sheds light on the development process. Such documentation, of false starts, intermediate steps, ideas accidentally gathered during ther course of other day to day activity, and personal recollections, is also the most fragile. Usually this kind of documentation 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 other conscious filtering process driven by other than historical retention aims. Therefore the time to undertake such an effort is now, rather than later, and much of the documentation will already be lost.

How quickly can such a program be mounted? Assuming the Computer Museum Board determines to proceed with establishing such a component of its collections, policies and procedures for such an archive could be drafted within four months. Plans could be made for special fund raising to assure adequate facilities and staffing for the project for its first two years and efforts to recruit an archivist could be launched in parallel with the drafting of policies and procedures. If no special facilities are required, a start-up date in the fall of 1987 would be a reasonable target.

Whether or not special facilities would be required by the

archives depends on decision made about the formats in which software code itself is to be maintained. If the recommendation of Appendix 1 is followed, ie. that original storage media be preserved as artifacts for exhibit purposes, but that the intellectual characteristics of code be stored on today's media, either eye-readable or readable by production computers which are being maintained for day to day operational purposes by the Computer Museum, then the facilities required by the software archive could be readied within the four months provided. They would consist of standard archives shelving with a wide range of archives boxes and tubes suited to the diverse formats of paper documentation, and of film, audio tape and photo archives containers. Software provided in machine readable form would be copied onto local DASD (and it would be worth looking into WORM drives for these purposes). Formatting data for communication to these devices would fall variously to the donors, if they had the facilities, or to a commercial service bureau. [It might be possible in some cases to borrow facilities of local universities and cultural institutions to make such transfers if they have the necessary equipment].

Needless to say, if the decision is made to retain some or all software in obsolete formats and to run it on the systems for which is was designed, the facilities required for the software archive will have to include a fully equipped temperature controlled and fire protected computer room (of potentially vast size) and substantial expansion space for the variety of drives, CPU's, and I/O devices which are represented in systems 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.

During the course of discussions, a number of programmatic decisions which the Computer Museum will face have been raised. Will the Museum decide to have an oral history program or other active documentation reconstruction effort? Does the Computer Museum wish to take on the role of a nexus of a network of institutions collecting software and exchanging information about it. If so, what implications does this have for publication and information systems development within the museum. Does the Computer Museum wish to assist other repositories by publishing guidelines for the development of software archives or training archivists to deal with software related issues. What role, if any, does the Computer Museum wish to take in the systematic collection of ephemera? - coordination or management of a sampling program? Finally, and this is the most important, what scope does the Computer Museum wish to set for its own collecting activity in order to encourage other institutions to collect, compete with them as little as possible, and still gather for the software archive an important record of the developments in the history of software?

APPENDIX 1

Is it necessary to be able to "run" the software?

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

Up front in our planning, therefore, we need to address the question of whether it is necessary to be able to run the software in a software archive. Our conclusion, I believe, will depend almost entirely on the answer to a prior question; What are the purposes of the archive? If the answer is public display or visitor education, then the software must run to be appreciated. Since the answer is to support scholarly research, 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 that research.

Let us define the issue tightly, so as to make certain that we are not setting up a straw man. There is no debate over whether research can be conducted on the commercial distribution 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 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 apprpriately 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 details of the code. Nor is there any debate that even though software documentation and advertising campaign records may be of value in understanding why one product succeeded where another failed, we will better appreciate consumer was resistance if we can "feel" the product and see it work. Finally, there is no dispute that published sources from the time and the correspondence of software company executives and their clients will point to the correct source of the 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?

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 particular 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 archive will often wish to acquire specific routines, or modules, even though the systems of which they are components are not historically important.

As is axiomatic among systems designers, 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 to achieve 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. Systems software specialists are notoriously brilliant, clever at applying fixes to their local systems, and equally poor at documenting them. The way the 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 to 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 machjine 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 distiction, 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.

APPENDIX 2

Legal Issues:

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 authoritiative 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, microcode 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 anwering the questions posed above.

As of the beginning of 1987, certain conclusions are warrented.

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 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 Associates 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, they 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 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 by

comparison of the resulting software in copyright and trade secret suits (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 softeware, but they also make it clear that when 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. TI settled out of court with Fujitsu and Sharp early in January 1987 under trerms 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 invcentions 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.

Finally, the courts have been extremely clear about what constitutes fair use of protected (copyright) materials. In this they have clarified for 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 a 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 and scholarly purposes, and technical criticism would all be permitted.

APPENDIX 3
ARCHIVISTS CONTACTED IN PHONE SURVEY

High Technology settings:
- Joan Warnow - AIP Center for History of Physics (Bookhaven)
- Vicki Davis - Lawrence Berkeley Laboratories
- David Baldwin - MITRE

Government Archives:
- Harlod Naugler & John McDonald - Public Archives of Canada
- John Fleckner - Smithsonian Institution

Computer Manufacturers:
- William Rofes & Robert Pokorak- IBM
- Bruce Bremmer - CBI

Professional Associations:
- Linda Resnik - American Society for Information Science
- Brian Kahin - EDUCOM

Universities:
- Maynard Brichford - University of Illinois
- Tom Hickerson - Cornell
- Helen Samuels - MIT
- Stanford

APPENDIX 4

Forms of Material for Documenting Software History

The process of developing, manufacturing, distributing and selling software is not fundamentally different from other research and development and marketting 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 Haas, Samuels & Simmons, Appraising the Records of Modern Science & Technology: A Guide (Cambridge, MIT, 1985). Unfortunately, Haas, Samuel & Simmons provide no guidance regarding software, and little with respect to processes driven by software. When we consider that most of the tools currently being reviewed in Science for use in 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 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. Once the Computer Museum determines what documentation of software and software history is most valuable for research, it should publish a corrective article or brochure to assist archivists to separate the issues of how best to retain data files for reuse and how best to document historical uses of computer based systems.

Software Development:

Software development does not differ greatly from other creative enterprises - it is essentially 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 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 their gropping progress towards the finished product.

Hardware/Software interaction effects:

Software makes generalized machines behave in specialized

ways. Obviously there are two ways to skin this cat - the other being 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 migrtating 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 of 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.

Educational contexts:

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, as illustrated by the immense programs operated by each of the major computing manufacturers to give computers to universities and the reliance of the government 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 their 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; alerting archivists to the value of these records for the history of software will increase the likelihood of their retention.

The Records of Software Protection:

While, ownership rights in software were poorly protected by each of the methods available: copyright, patents, code hiding and trade secrets until recently, 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 used this as a method of protection would also have had to construct files demonstrating how they invented/developed a technique which gave them a competitive advantage. These records, also found in corporate legal files, will be of future interest to historians.

The Software Business:

Since software emerged as a commodity it has been the subject of a massive ephemeral literature (in print and audio-visual formats) of advertisements, brochures, catalogs, demo disks, endorsements...etc. 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 everyday know that we are talking about immense volumes. Strategies for capturing these volumes need to be developed, whether they are cooperative collecting along divided lines of responsibility or random or periodic sampling.

Social organization:

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, these organizations should be documented. Some have already formed archives, the IEEE for instance. Others have elected archives; AFIPS is at the CBI. Still others are exploring it; ASIS is in negotiation with two potential repositories. Yet others will have

to be encouraged and cajoled by the Computer Museum and its colleagial institutions. Membership organization records are often thought to be baren, but committtees of the ACM, IEEE, ANSI, and other organizations have played critical roles in the development of standards which shape software, and provided for training and professional education. In addition they have testified before Congress, taken part in international conferences and trade delegations, and sponsored software exchanges amoung members and even the development of specialized software to meet shared needs.

FOOTNOTES:

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 assume if there was a better answer, they would have researched it, although I have a vague recollection that William Safire once was more precise...
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 error involved; cf.
   Adele Goldfine, 1946, trajectory calculation program for the ENIAC [reprinted article in Randell]
   Claude Shannon, 1938, logic program for his switching relay [Trans. Am. inst. Elect. Engin. 1938]
   R.E. Beard, 1942, actuarial table calculation programs for ? [J. Institute of actuaries, v.71, 1942 p,193-227]
   Howard Aiken, 1944, Manual of Operation of the Automatic Sequence Controlled Calculator, Harvard U.Computation Laboratory Annals, vol.1
   [reprinted by Babbage Inst.]
3) Mass produced is to be taken as a kind of production, not a volume: UNIVAC sold 15 UNIVAC I's. IBM manufactured 19 701's (the number sold is debated) and sold 15 IBM 702's.
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 Charles J. Bache, Lyle R. Johnson, John H.Palmer & Emerson W. Pugh, IBM's Early Computers, (Cambridge, MIT Press, 1986)
5) On Patrick see Moreau, op.cit #4; 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.
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.
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). Editors of the Harvard Business Review, The Digital Computer: Monster or Slave (Boston, HBR, 1955)

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 such innovations began to impact on daily life, it is startling to realize that the MIT Computation Center, the earliest computing facility at a university which was not dedicated to developing new computers but to using existing computers, was opened to researchers from 30 universities in June of 1957.

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

14) Science, December 19,1986

15) 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 Software (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 applications (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. Information Science Abstracts may prove to have a more useful scheme.

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

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

18) 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)

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

Archives & Museum Informatics
5600 Northumberland St.
Pittsburgh, PA 15217

January 22, 1987

Gwen Bell
Director
The Computer Museum,
300 Congress St.
Boston,MA 02210

Dear Gwen,

Enclosed is the report on the first phase of my study of software archives for the Computer Museum. I look forward to discussing it with you in Boston on February 6. Since we moved the schedule for the meeting and this report forward, I did not have the opportunity to present the findings in a systematic conclusion within the body of the report; I hope to present a set of concrete recommendations when we meet on February 6.

If you have any comments or suggestions to make prior to the meeting, I'd be happy to hear from you on 412-421-4638.

Sincerely yours,

David Bearman

I sent copies directly to Helen, Art Norberg, David Allison. Additional copies are for you to distribute. I should have a two page distillation of where we go from here done by the time you call me about this. We may have time to distribute it before the meeting.
Talk to you soon.