II. COLLECTING POLICY & SELECTION CRITERIA

A. CRITERIA BASED ON SOFTWARE CHARACTERISTICS

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

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

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

1. Function

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

2. Funding Source

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

---

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

3. Sponsor

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

4. Computing Environment

a) Hardware

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

b) Commmunication Environment

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

c) Software Environment

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

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

5. Distribution

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

6. Development approach

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

7. Ownership Restrictions

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

8. Design Theory

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

9. Business area

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

10. Social impact

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

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

B. CRITERIA BASED ON FORMS OF MATERIAL

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

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

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

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

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

C. CRITERIA BASED ON HISTORY OF SOFTWARE

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

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

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

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

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

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

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

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

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

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

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

D. CRITERIA BASED ON EVIDENTIARY VALUE

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

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

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

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

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