Iterations
An Interdisciplinary Journal of Software History

Preserving Software: Why and How

John G. Zabolitzky
ICOS Vision Systems GmbH

Date published: 13 September 2002

Information technology is the largest industry worldwide today, and is slated to be even more dominant in the future. The roots of this industry are hardware and software developments over the past 50 years. Future generations will find it hard to understand if these roots are not preserved today.

While a lot of hardware artifacts have been well preserved, I strongly suspect that some past software artifacts have already been destroyed irrecoverably. (Where are the early UNIVAC operating systems? The first FORTRAN/COBOL/ALGOL compilers?) There is a grave danger that in the near future large bodies of historically relevant software will be destroyed. Now is the time, therefore, to take action and preserve these objects for future study.

Why

It has been argued that no working hardware of bygone times is available or will be available, and therefore the preservation of software does not make any sense. This statement is incorrect in every respect, as I will show.

First, it is simply not true that no historic working hardware is available. There are several examples, even of the rather fragile vacuum-tube type:

- IBM 650 at IBM Haus zur Geschichte der Datenverarbeitung, Sindelfingen, Germany, and several later IBM machines at the same location.
- IBM 700 series machines in the collection of Paul Pierce, in Oregon, USA.
- Control Data and Cray machines in my own workshop (see www.cray-cyber.org).

- Hundreds of early Digital Equipment machines in the possession of smaller organizations and individual collectors (true for other minicomputer manufacturers as well).
- Thousands of first-generation personal computers in various private collections and at smaller organizations.
- The Computer History Museum at Moffet Field, Mountain View, California, is working on restoring a number of systems; there is no doubt about their future success.
- There are also commercial operations known to operate 20+ year old equipment, because it is in some cases cheaper than rewriting the software.

Conspicuously, the large, renowned public museums of the world are absent from this list. This is simply a matter of allocation of effort, financially as well as creatively. Restoration and continued operation of historic computing equipment requires special knowledge and skills unique to this task. In all of the above cases volunteers in possession of such knowledge and skills have done the actual work, and are currently training younger interested people so that knowledge and skills will be preserved. It is nothing but a matter of decision making to establish similar operations at any of the large institutions; with their clout they will easily attract volunteers to do the work, if it is only well publicized. The present state, however, is a more distributed effort by specialists, see list above.

I must conclude that hardware will be preserved and operational in the future—if not at the large institutions then at distributed and specialized locations. It will be possible to run historic software on these machines. The meaningful operation of these machines will require the possession of an operating system, programming system, and application software, at least in binary object code format.

Second, the task of hardware preservation is significantly simplified if operating system software is available in source code format.

The major complication for preserving, restoring and operating historic computers results from the moving parts subject to mechanical wear—NOT from the electronic parts (with perhaps the exception of short-lived vacuum tubes). The encapsulated semiconductors, passive components, circuit boards, connectors, and wiring making up the processing units are essentially free of natural degradation, assuming that the atmosphere is non-corrosive. Stocking of spare parts is possible; in many cases, parts of modern manufacture may be used.

The problems arise from magnetic disks, tapes, mechanical card/tape readers/ punches, and generally all peripheral subsystems with mechanical parts. Magnetic surfaces, positioning mechanisms, transport mechanisms are susceptible to degradation due to normal operation. However, the processing of information—which is the core of preserving a historical operation—may be kept up by substituting modern peripherals for the historic ones. For example, a historic disk controller and disk drive could be replaced by a specially designed new disk controller and modern hard disk drives. This has been done in some cases, and is a limited volume of work. This can be done one hundred percent software compatible with the original disk subsystem, and moves disk reliability up to modern standards, very significantly reducing the maintenance effort on the historic machine. Since the historic drives still may be powered up and will provide the historic look-feel-and-sound, this replacement is neutral to the observer.

This kind of operational replacement is simplified if it is not transparent to the operating system software. If that software is accessible in source code format, and not only in binary object code format, it could be modified for the convenience of the new hardware add-on designer.

Logical operation, i.e., information processing on essentially historic machines, will be possible on an even larger scale than for the case of pure historic machine configurations. There is a mutual reinforcement: The more software that is available, the more hardware can be kept running, and the more hardware that is running, the more software is worthwhile to preserve.

This route has in fact been followed at some commercial installations. Some companies have chosen to keep their software investment valid, avoiding the writing of new software for an existing application by upgrading old computers with new hardware.

Third, simulators for a number of machines have been built, and can be and will be built in the future. Historic software can be run on such simulators. On the user level, assuming some dialog on some terminal equipment, there need not be any perceptible difference between simulated operation and software running on historic hardware.

This method of operation can be carried further: a simulator can be interfaced to any piece of historical hardware desired to be revitalized. For example, the operator's console of a historic machine could be physically interfaced to the simulator. The same is true for any peripheral. Simulators will run the binary object code. However, interfacing to newer peripherals will be significantly simplified if the source code is available.

Anything from pure simulation to full complement of historic peripheral operation can be realized with the use of simulators. The experience of the original user sitting at a terminal or console can be one hundred percent recreated in this way, provided the software is available.

This approach is in fact similar to and related to emulation of the IBM 1401 on the more modern IBM System/360s and beyond, which has been, and maybe still is, used widely.

Fourth, the evolution of software methods, techniques, styles, etc., is described in many books and articles. However, all of that is essentially hearsay: what *actually* has been done (and what may be different from what the active players in this area may report since they might have wished to do something different) can only be discerned and proven by examining the source code. The source code of any piece of software is the only original, the only artifact containing the full information. Everything else is an inferior copy.

The binary object code format of software is of course much more abundant than the source code format. Any installation of any computer system requires the possession of the binary object code for the operating system, the programming system, and the applications. Only the manufacturers of the software, be it the computer manufacturer or some software company, will hold the source code format. Unfortunately, the process of generating the object from the source obliterates an enormous amount of information present in the source code. How was this actually accomplished? What methods and techniques were used? Why could Company X be so many times more efficient than Company Y? These questions can reliably be answered only if one is in possession of the source code, the only proof of what actually was done.

There can be no doubt that the distributed community of smaller organizations and private collectors is in collective possession of a very large amount of object code. However, source code of sensitive and historically important software cannot be expected to be found there (e.g., the IBM System 360 Operating System and Digital Equipment VAX system software).

The preservation of source code format of historically important software, which is mostly owned by large corporations, can be dealt with only by renowned organizations carrying the clout to impress these companies.

This is important because only the simpler, lower level functionality of any information processing system is implemented in hardware. The more complex features making up the functionality as well as the look-and-feel of such systems are implemented in software. Preserving such software is the only means of preserving the experience of twentieth-century information processing for future generations.

While the object code in conjunction with actual hardware or simulators can recreate the look-and-feel, insight into the actual implementation work can be gained only by looking at the source code. The source code can provide the understanding of “how did they do that in those days?” Furthermore, possession of the source code simplifies the task of keeping old hardware running.

Fifth, we must consider the audience. It has been asked, “Who will look at past software in the future?” There are several answers to this question:

- The IT industry being the largest today and expected to grow further will force a growing fraction of the population to be not only computer-literate, but also software-literate. The number of software developers capable of reading source code is constantly growing, not only in absolute terms but also as a percentage of the entire population. Some of these individuals in the future will want to know how things were done in the past.
- There is a lot of arbitrariness in software; most things can be done in many different ways. Future historians might want to follow the evolutionary paths of software methods or techniques: Who has used what technique at which point in time? Who was influenced by what? The source code is the only proof.
- Large and small organizations as well as private individuals operate historic hardware or simulators. It is a growing community.
- How can we know what kind of questions future generations will ask? We should provide the basic artifact such that these people will be able to find out whatever they may want.

Finally, why should software be preserved in object code as well as in source code format? This is the only way to allow future generations access to two dimensions of past and current information processing:

- The sensual experience of walking into a mainframe computer room, the combination of air movement and smell, vibrations through acoustics as well as the floor, noise of working peripherals, and optical perception of console lights and screens; *only with software can computers be shown with lively personalities as opposed to dead corpses.*
- The intellectual study of software evolution as documented within the source code as the only original artifact of what actually was done.

How

There is a concern that preservation of software is technically difficult because of the deterioration of media. In fact, the opposite is true. If conceived of in the proper way preservation of software is a rather simple task—getting hold of the software is the difficult part.

The important notion is that software is not a physical artifact. Software consists of bits, and not of atoms. While it may be nice to preserve actual media like half-inch tapes, floppy disks, etc., this is by no means central. The preservation of media is a task in its own, and is entirely independent from the preservation of software. As is well known, media in fact need regular copying in order to keep the information on them valid—be it software or anything else. This requires operation of the appropriate copying hardware. Obviously, wherever actual historic hardware is in operation, the operators of that hardware will see to the copying of their media in the normal way of making backups, in the interest of continued operation. The preservation of information on historic media is therefore to be viewed as a small and necessary part of operating the corresponding drives.

Preservation of software is identical to preservation of information. The physical hardware used and the media used are quite irrelevant. Conceive of a piece of software simply as some computer file, to be stored on any computer of choice for convenience. The "software archive" now is a directory tree on some computer. This computer at the same time is connected within the preserving organizations computer network, like other file servers, and all (or most) of the organizations' computers. The normal, everyday file backup procedures of that organization then may be extended to the "software archive" as well—whoever is doing the backup of the other data of the organization, e.g., payroll data, databases (archives of physical artifacts) etc., without much additional work can do that backup as well. The historic software therefore lives on standard, normal, modern hard disk drives of some standard file server computer, and is regularly backed up to tape, using the normal operating procedures of the organization's IT department.

In the course of time file servers will be modernized. In the same way as other information was easily moved from one generation of equipment to the next generation, it can safely be assumed that the same will be true in the future. When a new generation of file servers is installed, all information needs to be transferred—including the historic software archive. When backup media is changed in the future, no problem will arise. In the same way that the lives of current data files of any organization are essentially infinite, this will also be true for the software archive. If at some future point in time a totally different computer architecture grows to reign, it is perfectly safe to assume that the proper data migration tools will be provided—simply because everyone on the planet will need them. Therefore, once we have the software archive within a standard file system, the task of preservation is solved.

In this way it is also trivial to make the software publicly accessible. Where allowed, the file server could be the Internet FTP server of the organization. The directory structure serves as the catalog. Of course, additional cataloging would be required to facilitate search operations.

The only remaining point is to transfer the software from whatever media it may be in, to some standard file format on current file server systems. There are a number of topics to be discussed here:

1. physical procedure
2. data formats for binary objects as well as source code formats
3. directory structures and aggregation

1. Physical procedure. The software to be preserved will come on some more or less standard or outmoded media. For all of these, however, current commercial

solutions exist to transfer the data (bitstrings representing the software) onto current, modern media.

> Examples:
>
> *Half-inch tape.* There exist half-inch magnetic tape drives which can be attached to PC-type computers. A software archiving station would consist of such a tape drive, attached to some standard PC, networked to the software archive file server. In the case of older tapes, which may have deteriorated, some expertise and tools for data reconstruction may be required. Commercial services also exist which will do the entire data transfer.
>
> *Floppy disks.* Rather old floppy disk drives may still be attached to modern PC-type computers; there exist universal reading software products which will read non-Microsoft disk formats. More convenient may be reading the disks on their native computers, assuming some such can be found, and transferring the data via common exchange media, or some communication (RS-232) or networking method.
>
> *Disk packs.* These would require an operational computer and data communications or exchange media. Most machines with disk packs also would allow copying to half-inch tape, which then would be processed as outlined above.
>
> Other magnetic tape formats can be handled in much the same way.

2. Data formats. This item requires a little more consideration since it is related to the anticipated usage of the data, i.e. binary or source software.

The binary would be useful only for running on historic hardware, or else on a simulator. Therefore, just mapping tape blocks or disk blocks to a modern file format would do. The extraction of the data to whatever format the future user might want it to be in would be the inverse of one of the methods above.

For source code software two different potential usages exist: human reading for investigations into software methods, etc., or compilation (possibly after modification) on historic hardware or a simulator. For both applications program source code format of the file server system would be appropriate. This might require some small data extraction program within section (1) above. For the second application, alternatively tape or disk blocks of the target (historic) system would be useful as well. Since both formats carry all the information in principle, they can be transformed into each other by a suitable reformatting program. It is a mere matter of convenience if both formats or only one of the two is offered on the file server.

3. Directory structures and aggregation. It should be fairly simple to decide on a case-by-case basis, probably starting with some manufacturer at the top level ("IBM," "CDC," "MS," etc.), descending into machine families where appropriate, from there descending into the maybe several operating and programming systems and applications.

None of these tasks is new. The entire set of procedures has been carried out repeatedly, on smaller and on larger scales, by private collectors and small institutions. The expertise exists and can be tapped when and where required.

What To Do

URGENT: Because of lack of concern, motivated by tightness of space, corporations will destroy and have destroyed in the past important source code programs. While object code may still exist in some places, this is not the original as explained above. The prevention of further destruction should be a primary and urgent goal. This primary goal should be the acquisition of source code format software from the relevant owners. The first step might be the creation of a list of "national heritage software," which one may then seek to acquire.

Passive collection, as opposed to the above active strategy, will most probably result in a set of software which has only little relevance. Important pieces will be missing, and pieces of lesser importance will fill up space.

In order to make the best use of it, the following items optimally should be found in conjunction with a given piece of software, and all cataloged with reference to the same short-hand title:

- source code in human readable format
- source code in target machine disk/tape blocks
- object code in target machine disk/tape blocks
- user manual(s) for this software
- installation and maintenance manual(s), where appropriate
- design documents for this software (probably the most difficult to find, but also the most valuable to have)
- other documentation, like user guides, books, tutorials, reports on that software
- example run, input and output (frequently contained in the user manual)

For larger programming/operating systems each of these may be an entire set, in the case of the software items a directory tree.

John G. Zabolitzky, "Preserving Software: Why and How," *Iterations: An Interdisciplinary Journal of Software History* 1 (September 13, 2002): 1-8.