PRESERVING.EXE

Shelved software from the Library of Congress collection at the National Audio-Visual Conservation Center.

October, 2013

Toward a National Strategy for Software Preservation

A report from the National Digital Information Infrastructure and Preservation Program of the Library of Congress, focused on identifying valuable and at-risk software. Topics covered include executable software preservation, game preservation, electronic literature and ideas for approaches to ensure long-term access.

Contents

Preserving.exe

TOWARD A NATIONAL STRATEGY FOR SOFTWARE PRESERVATION

America runs on software. From operating streetlights and financial markets, to producing music and film, to conducting research and scholarship in the sciences and the humanities, software shapes and structures our lives. Software is simultaneously a baseline infrastructure, and a mode of creative expression. It is both the key to accessing and making sense of digital objects and an increasingly important historical artifact in its own right. When historians write the social, political, economic and cultural history of the 21st century they will need to consult the software of the times.

On May 20-21 2013, the National Digital Information Infrastructure and Preservation hosted "Preserving.exe: Toward a National Strategy for Preserving software" a summit focused on meeting the challenge of collecting and preserving software. The event brought together software creators, representatives from source code repositories, curators and archivists working on collecting and preserving software, and scholars studying software and source code as cultural, historical and scientific artifacts.

The goals of the summit were as follows:

- Articulate the problems and opportunities of software preservation.
- Encourage new partnerships and collaborations that will support organizational roles and responsibilities related to software preservation.
- Advise the Library of Congress on next steps, including different methods and approaches that will develop criteria for assessing long-term value, and new organizational models that best support the stewardship of software content.

This report is intended to highlight the issues and concerns raised at the summit and identify key next steps for ensuring long term access to software. To best represent the distinct perspectives involved in the summit this report is not an aggregate overview. Instead, the report includes three perspective pieces; a curatorial perspective, a perspective from a humanities scholar and the perspective of two scientists working to ensure access to scientific source code.

Henry Lowood, Curator for History of Science & Technology Collections at Stanford University Libraries, describes three lures of software preservation in exploring issues around description, metadata creation, access and delivery mechanisms for software collections.

Matthew Kirschenbaum, Associate Professor in the Department of English at the University of Maryland and Associate Director of the Maryland Institute for Technology in the Humanities, articulates the value of the record of software to a range of constituencies and offers a call to action to develop a national software registry modeled on the national film registry.

Alice Allen, primary editor of the Astrophysics Source Code Library and Peter Teuben, University of Maryland Astronomy Department, offer a commentary on how the summit has helped them think in a longer time frame about the value of astrophysics source codes.

For further context, we have included two interviews that were shared as pre-reading with participants in the summit. The interview with Doug White explains the process and design of the National Institute for Standards and Technology's National Software Reference Library. The NSRL is both a path breaking model for other software preservation projects and already a key player in the kinds of partnerships that are making software preservation happen. The interview with Michael Mansfield, an associate curator of film and media arts at the Smithsonian American Art Museum, explores how issues in software preservation manifest in the curation of artwork.

The term "toward" in the title of this report is important. This is not a national strategy. It is an attempt to advance the national conversation about collecting and preserving software. Far from providing a final word, the goal of this collection of perspectives is to broaden and deepen the dialog on software preservation with the wider community of cultural heritage organizations. As preserving and providing access to software becomes an increasingly larger part of the work of libraries, archives and museums, it is critical that organizations recognize and meet the distinct needs of their local users. In bringing together, hosting, and reporting out on events like this it is our hope that we can promote a collaborative approach to building a distributed national software collection.

The Lures of Software Preservation

HENRY LOWOOD, CURATOR FOR HISTORY OF SCIENCE & TECHNOLOGY COLLECTIONS AND FILM & MEDIA COLLECTIONS, STANFORD UNIVERSITY LIBRARIES

INTRODUCTION: LIFE-CYCLES AND WORKFLOWS

Libraries, archivists, curators and others working in the realm of digital preservation have become accustomed to thinking about the long-term care of bits in ways that emphasize activity rather than storage. Metaphors such as "life-cycle management" and "workflows" express both an understanding of the multi-faceted nature of the process and a commitment to the lifetime -- or more accurately, lifetimes - of effort involved. The most influential statement of this approach has been the Curation Lifecycle Model of the Digital Curation Centre. The DCC describes its model as a "graphical, high-level overview of the stages required for successful curation and preservation of data from initial conceptualization or receipt through the iterative curation cycle." Selection, ingest and storage generally are one-time actions that occur in a fixed sequence, while assignment of metadata, tools development and administrative actions such as auditing and migration are likely to be repeated throughout the life-cycle. Thus the model encompasses both a linear sequence of work stages and the cyclic repetition of specific tasks.

Metaphors of life-cycle and workflow bring the continuous nature of institutional commitments to software preservation into focus. These commitments are long-term and repetitive, and they require planning and infrastructure. From the perspective of digital curation understood as data preservation, software preservation fits more or less neatly into the model offered by the DCC and the workflows developed for ingest, storage, and auditing of digital objects. For a collection curator such as myself, recent progress towards long-term preservation solutions has been both necessary and exciting. Equally important aspects of collection management around access to and delivery of digital assets have received relatively sparse attention. How researchers will actually put their hands and eyes on historical software and data collections generally has been bracketed out of data curation models focused on preservation. The National Archives of Australia has described digital preservation as a "four step process" leading from "manifest" (selection of records) through preservation and storage. The process focuses on integrity and auditing of these records, but the description of it gives no indication that the authors considered likely use scenarios or issues of access to them. The DCC locates the life-cycle action identified as "access, use and reuse" in the sequential workflow between "store" and "transform." A "curation checklist" for this action provides general guidance: Know what you expect users to do with the data, understand and communicate the data's significant properties, formulate and enforce restrictions on use, and provide contexts for search and use. Yet, the DCC also provides little in the way of specific information about the development of use cases and how to account for them. Similarly, Ross Harvey's authoritative *Digital Curation: A How-To-Do-It Manual* (New York and London: Neil Schuman, 2004) acknowledges the need to manage data in a fashion that accounts for

designated user communities, but without offering cases of discovery, use and access. The motto of the DCC hints at the rationale for limiting the life-cycle model to data preservation activities: "Because good research needs good data."

This essay is not a critique of digital curation. The work described above is the conceptual bedrock upon which institutional data stores and repositories, preservation activities and digital archiving programs have been constructed. The accomplishments are adding up as capacity and functionality is developed, tested, and switched on-line. Rather than as criticism, think of this essay as a reminder that we still have a few unchecked boxes on our to-do list: description, metadata creation, access and delivery mechanisms. I am merely stating the obvious. Success in providing good data means that we must account for good research that (and good researchers who) will use these data. Thinking about how digital collections will support research in software history is a good place to start.

SOFTWARE USE CASES

So, how can we productively shift some of our attention to access and use? A good place to begin is use cases. The term historically has been closely tied to software development. Lore has it - and an analysis using Google's Ngram Viewer supports the claim - that its consistent use begins with Ivar Jacobson's *Object-Oriented Software Engineering: A Use Case Driven Approach* (New York: ACM, 1992). According to Alistair Cockburn, a key popularizer of use cases, Jacobson began using the term by the mid-1980s to describe his approach to software engineering. In “Use Cases, Ten Years Later” (STQE Magazine, March/April 2002), Cockburn defines the use case as "a prose description of a system’s behavior when interacting with the outside world." In other words, the use case tells the software designer what happens when a particular kind of user interacts with a software system. It is, as Cockburn emphasizes, a "scenario of behavior."

Jeff Rothenberg gave us the Mother of All Use Cases for software preservation in "Ensuring the Longevity of Digital Documents," the influential article he published in *Scientific America* in 1995:

"The year is 2045, and my grandchildren (as yet unborn) are exploring the attic of my house (as yet unbought). They find a letter dated 1995 and a CD-ROM. The letter says the disk contains a document that provides the key to obtaining my fortune (as yet unearned). My grandchildren are understandably excited, but they have never before seen a CD—except in old movies. Even if they can find a suitable disk drive, how will they run the software necessary to interpret what is on the disk? How can they read my obsolete digital document?" (Jan. 1995, 42.)

Note that Rothenberg shifts the meaning of "use case" by profiling user need rather than the user's interaction with a system. It is a use scenario. Rothenberg's scenario introduced readers to the problem of preserving digital files in a manner that future generations would find usable. His essay deserves credit for opening the conversation on data migration, encoding of bit streams, and preservation of representation and contextual information. It is instructive to note as well that the conclusions Rothenberg drew in his article were based on specific ideas about how digital documents would be used. If word processing documents were reduced to text, what would be the implications of lost formatting instructions? If the functioning of software programs could not be replicated, what would be the effects on the rendering of document files? While Rothenberg focused on "reading" documents encoded in bit streams,

we can of course imagine similar potential needs for other kinds of historical data and software. As repositories of historical software plan for access, use cases both as scenarios and descriptions of system behaviors will shape expectations about the technologies and services required to deliver digital collections to researchers.

Since the mid-1980s, I have been involved through a succession of activities in the history of computing, development of historical archives and collections to document that history, and digital game studies. Specific milestones in the progression of my thinking about historical software collections include the first conference on the topic at the Arden Conference Center around 1990, the National Software Archives group led by David Allison of the Smithsonian Institution in the mid-1990s, and the Software Task Force of the Charles Babbage Institute, which issued its Final Report in 1998. The How They Got Game project at Stanford - which I have led or co-led since 2000 - participated in both of the two Preserving Virtual Worlds (PVW) projects funded by the Library of Congress and Institute for Museum and Library Services, respectively, from 2007 through 2012. With PVW, the leading edge moved from archival records to preservation of digital objects such as software. In retrospect, it seems to me that PVW followed the sensible trajectory of problems that led from scoping work on the description, selection and ingest of game software, electronic literature, and virtual world data to an investigation of how better understanding of significant properties might inform preservation, management, and auditing of software collections.

While these projects were underway, the Stanford Libraries (where I work) participated in a string of digital library projects (including AIMS and several NDIIPP projects) and launched new services, such as the Stanford Digital Repository and our Forensics Lab. In short, the pace of work in digital repository development has been accelerating, especially over the past decade. Stanford currently is involved in a partnership with the National Institute of Standards and Technology to migrate data from our largest collection of historical software: The Stephen M. Cabrinety Collection in the History of Microcomputing.

Connecting the dots from archives of software history to the PVW work and the progress in repository development has led me and several of my colleagues to conclude that we have reached a high level of competence in grabbing and storing data, but we have not accomplished nearly as much to create an infrastructure, workflow or policies for providing access to these data. For example, we lack metadata schema, terminology and ontology for interactive software, we have many issues to work out with respect to rights management, and there is still work to be done in evaluating the implications of technical options for executing historical software.

It is time to begin the conversation about access. Of course, we have the good fortune of having at our disposal technology, workflows and capacity for capturing, storing and migrating digital materials. Repositories have been launched and significant collections have been ingested by them. It is encouraging to see so much activity on one end of the life-cycle. In thinking about the other end, I have kept three points in mind about building systems for handling access: (1) How we plan for these systems will be guided by an understanding of use cases and the paradigms that we build out of these scenarios; (2) traditional curation activities (selection, collection, and provision of access to software and other digital objects) is deeply invested in the new digital curation (ingest, preservation, auditing, management, tools development); and (3) scenarios for research use of software libraries must take account of archival documentation that provides context and meaning for digital libraries.

USE CASES AND THE THREE LURES

I would love to give you a beautifully developed set of compelling use cases. Unfortunately, such a set does not exist. Certainly, the projects mentioned earlier, my own research and collection uses with which I am familiar provide a bunch of use examples, but the sum of these experiences is more anecdotal than systematic; again, we are still in the early days of access to digital collections. Instead of claiming to deliver the canonical use cases, I will offer a few thoughts about potential pitfalls for the imagining of these scenarios. The idea is not to criticize work that has taken place. It is too early for that. In fact, it would thrill me if my concerns turn out to be straw men, meaning that when we look backwards in a few years, we will see that these problems never existed.

My first inclination was to call these pitfalls "sirens" or "siren songs." The sirens, of course, were mythical creatures whose singing lured ancient seamen to their death. The mental image of Odysseus preparing his crew to resist these temptations has some appeal as a metaphor for certain ideas about playback and experience of historical software that might lead digital repositories astray. However, the associative baggage of connecting this metaphor to notions of feminine seduction and deception is not appealing, particularly when writing about subjects with as many historical issues around inclusivity as digital games and computer programming. Instead, I will refer to the *lures* of historical software use. A lure, according to the *Oxford Dictionary,* is "something that tempts or is used to tempt a person or animal to do something." It is allied with words like trap and bait and thus fits nicely with the previously employed "pitfall." Problem solved. The three lures I have in mind are "the screen," "the authentic experience" and "the executable." Tempting ideas about access to and use of historical software collections have been constructed around each of them.

The Lure of the Screen

The lure of the screen is related to what media studies scholars such as Nick Montfort, Mark Sample and Matt Kirschenbaum have dubbed (in various but related contexts) "screen essentialism." In an extremely general and simplistic form screen essentialism means that what counts in digital media is what is on the screen. With respect to software preservation, the more general idea is that the significant properties of software are surface properties. These are properties that we perceive as being generated by our interaction with software: graphics, audio, responses to our use of controllers and so on. Besides being the properties that users perceive and manipulate, they are arguably the properties that designers consider as the focus of user interaction and the properties that are easiest to inspect and verify in a repository if software executes successfully.

The second Preserving Virtual Worlds project was concerned primarily with identifying significant properties of interactive game software. The potential value of significant properties research for digital media preservation lies in its application to tasks ranging from strategic planning to auditing whether changes have occurred in digital objects over time. According to the InSPECT report (Knight, 2009), these properties tell us which characteristics of digital objects "must be preserved over time in order to ensure the continued accessibility, usability, and meaning of the objects, and their capacity to be accepted as evidence of what they purport to record." On the basis of several case sets and interviews with developers and other stake-holders, PVW2 came to the conclusion that isolating surface properties such as image colorspace, while significant for other media such as static images, was not a particularly useful approach to take for game software. With interactive software, significance appears to be variable and contextual, as one would expect from a medium in

which content is expressed through a mixture of design and play, procedurality and emergence. With respect to surface properties in particular, software abstraction levels are not “visible” on the surface of play. It is difficult if not impossible to identify and monitor significant procedural aspects of game design and mechanics, programming and technology simply by inspecting properties expressed on the screen.

As with other media, it is unlikely that software will be preserved without changes in interactivity and appearance, which is why we try to understand significant properties. It turns out to be quite difficult to predict which characteristics will be considered significant by future researchers. An alternative or companion approach for auditing the operation of a historical software program might be to verify the execution of data files. The integrity of the original software can always be evaluated by comparison to documented disk images or file signatures such as hashes or checksums. However, when data migration or delivery environments call for changes to the software, we need to evaluate the performance of the resulting code. Instead of asking whether the software "looks right," we might first test whether it runs verified data-sets that meet the specifications of the original software. Examples of the external files that software is expected to accept as data range from word processing documents to saved game and replay files. Of course, visual inspection of the content will sometimes be necessary to verify execution by the software engine; failure will not always be indicated by crashes or error messages, but may only be evident when formatting does not look right or audio will not play. Questioning screen essentialism is not a complete rejection of the relevance of surface properties for preservation activities. Nevertheless, I expect that workflows built around testing of data execution will be more scalable and more easily assigned to machines than a system that relies on analysis of what is seen on the screen.

The Lure of the Authentic Experience

Everyone in game or software studies has worked with emulators or at least has read or heard about emulation (including display emulation), virtualization, or physical reconstruction of components such as controllers and cabinets. Many if not most emulation projects struggle to recreate an authentic experience of operating a piece of software or playing a digital game. The yardstick for measuring authenticity is generally taken to be the experience of using historical hardware and software. Note that this measure is in sympathy with an emphasis on surface characteristics as significant properties. The Lure of the Authentic Experience then is a mandate that digital repositories at minimum must be careful not to preserve digital objects in a manner that would interfere with the production of such experiences. At maximum, repositories would be expected to provide "reading room" access models that deliver these experiences on-site. In the minimum case, the repository might need to collect hardware specifications, drivers or support programs that account for interfaces between, say, game software and original hardware components such as displays, and controllers. If these are lost, it may be difficult or even impossible to reconstruct the information needed to provide a reconstruction of the historical look-and-feel of a program. In the maximum case, the repository designs and builds authentic access environments and provides access to them - a service that probably requires that researchers travel to the repository to use historical or bespoke hardware. Some notion of authenticity exists for any medium, of course. In some cases, it focuses on objects (such as the difference between a rare manuscript and a modern reproduction of it) and in others on environments in which the medium is experienced (such as the difference between viewing a movie in a theater or at home on YouTube).

Debunking the Lure of the Authentic Experience is not a comfortable thing to do. Authenticity is a concept fraught with emotional and intellectual issues, ranging from nostalgia and fandom to disagreements about essential methodologies in fields such as history of technology, media archaeology, or critical media studies. It is a minefield, but I will try to sum up a few of the issues without stepping on too many mines. The first problem is perhaps an academic point, but nonetheless important: Authenticity is not given to us, it is constructed. Put another way, whose actual experience counts as "authentic" and how is it documented? Is the best source a developer's design notes? The memory of someone who used the software at the time it was released? A marketing video issued by the publisher? The researcher's self-reflexive use in a library or museum? In the case of software with a long history, such as the game *Breakout* or the application program *Microsoft Word,* how do we account for the fact that the software was used on a variety of platforms - do repositories have to account for all of them? And what do we do if sources conflict or the software is capable of being used in a variety of ways? For example, we might consider that an authentic way of playing a multiplayer "death match" with the early first-person shooter *DOOM* would require peer-to-peer networking using a local area network, a mouse-and-keyboard control configuration and a CRT display. In fact, there are documented cases of competitive play with different arrangements of hardware: track-balls, programs or ports that enabled Internet-based multiplayer via TCP-IP, monitors of various shapes and sizes, and so on. Which differences matter? We can extend this logic to home computers and game consoles of the 1980s - every television was not a color television, for example.

Another problem with the Authentic Experience is that often it is just not that useful to the researcher; it may not be the most useful way to understand historical software in execution. Nick Montfort showed attendees at the preserving.exe conference that even when an emulated version of a software program is not used with historically authentic display and control hardware, the emulator interface compensates by offering real-time information about system states and code execution. The trade-off for losing the experience is that fresh insights might emerge from data about the underlying mechanisms of the software once they are made visible. We might generalize from such observations to ask what kinds of questions researchers such as historians of technology, practitioners of code studies or game scholars are likely to ask about historical software. While I do not have a crystal ball at hand, my guess is that for most researchers documentation of historical use of software is a more valid source than a personal experience somehow deemed authentic. This is not to say that engagement with artifacts - both physical and "virtual" - will not be a necessary part of the research process. It is difficult to imagine how a full appreciation of historical software design could be gained without some appreciation for the constraints and affordances of original hardware, for example. It seems likely that access to original technology will have a role to play in gaining that appreciation. Surprisingly perhaps, some studies (Margaret Hedstrom's work, in particular) suggest that current users prefer reworked or updated experiences of historical software. There is also anecdotal evidence from museum and other exhibitions that visitors who did not grow up with historical computer technology prefer to use and experience updated versions. In light of these various observations, it is difficult to argue that a digital repository should be preoccupied with delivery of the Authentic Experience as part of its core mission. A better use of limited resources would be to insure that validated software artifacts and associated contextual information are available to researchers who are inclined to do this work on a case-by-case basis. Put bluntly, digital repositories should consider the Authentic Experience as more of a reference-point than a deliverable, as a research problem rather than a repository problem.

The Lure of the Executable

An attractive version of the end-product of software preservation is the library of executable historical software. The work to reach that product involves a series of tasks from selection and collection through ingest and migration from original media and creation of technical and rights metadata, as well as provenance, descriptive and contextual information. This workflow is at the heart of the life-cycle model. While the model says little about related physical components such as packaging, manuals and box inserts, this can be considered as a separate archiving problem and as projects at the Strong Museum, University of Texas, Stanford University and elsewhere indicate, this work is underway. The point for software preservation is that the Lure of the Executable compels digital repositories to focus without distraction on building collections of verified historical software that can be executed on-demand by researchers. This has been the Holy Grail of digital curation with respect to software history.

What could possibly be wrong with this mission, assuming that it can be executed? As I have argued on other occasions there are at least three problems with focusing primarily on the software library. Fortunately, they are problems of omission, rather than commission. The first problem is that software does not tell the user very much about how it has previously been used. In the best case, such as previously installed software available in its original use environment (not a common occurrence), application software might display a record of files created by users, such as a list of recently opened files found in many productivity titles like Microsoft Office. The more typical case for a software library however will be that software is freshly installed from the library and thus completely lacking any information about historical use.

The second point might best be illustrated through the example of virtual world software, such as *Second Life* or a game world such as *World of Warcraft*. Let us assume that we can capture every bit of historical data from a virtual world server and then successfully synchronize all of this data with carefully installed (and patched) software, we can reverse engineer authentication controls, and we can solve the complex problems of ownership and rights. We will have accomplished an act of perfect software and data capture, and of course we will then preserve all of the associated digital objects. Now we can create a historical time machine through which we can fly through the virtual world at any moment in time of its existence. Of course, historians would applaud this effort, until it becomes evident that all we can do is watch. In the absence of historical documentation that helps the researcher to understand motivations and reactions of the participants, the pay-off will be limited. Content without context.

The third point and perhaps the most fundamental is that the documentation that is a prerequisite for future historical studies of software and digital media such as games and virtual worlds is simply not located in software. It is, in a sense, on both sides of software: the design materials (including source code) that document software development and the archives, both digital and non-digital, that document context, use and reception. It is important to understand that this is not just a problem for historical research. It is also a problem for repositories, whether they are doing the work of digital preservation or addressing the needs of a researcher requesting access to this software. If contextual information such as software dependencies or descriptions of relationships among objects is not available to the repository and all the retired software engineers who knew the software inside-and-out are gone – it may be impossible to get old software to run.

In *Best Before: Videogames, Supersession and Obsolescence,* James Newman argues that a host of publishing, retail, marketing, journalistic and other practices have cultivated a situation "in which the new is decisively privileged and promoted and the old is constructed either as a benchmark by which to measure the progress of the current and forthcoming 'generation' or as a comprehensively worn out, obsolete anachronism to be supplanted by its update, superior remake or replacement." (p. 121). Newman is not optimistic about software preservation. This does not mean that he is pessimistic about every possibility for historical preservation, however. In a section of his book provocatively called "Let Videogames Die," he argues that a documentary approach to gameplay might be a more pragmatic enterprise than the effort to preserve playable games. It represents a "shift away from conceiving of play as the outcome of preservation to a position that acknowledges play as an indivisible part of the object of preservation." (p. 160) Stated more generally, software preservation is not simply about preserving historical software for use as a research act, but also about preservation of documentation about historical use. This does not mean that the Lure of the Executable is a false idol, but rather that the Library of Executable Software is only a partial solution to the problem of software preservation.

CONCLUSION: IS RESISTANCE FUTILE?

At the end of the day - or more likely, at the end of several thousand more days - repositories may find it difficult to resist the three lures. After all, the purpose of providing access to historical software collections will be to enable researchers to understand how software worked, the intentions of software designers, and the affordances and experiences offered to software users. Of course, lures are called lures for a reason: They are shiny and attractive, and they reel us in. Moreover, the lures I have identified are not misguided goals. Obviously, there will be situations in which repositories should make it possible for researchers and other users to make an accurate assessment of surface characteristics, run the software, experience using it, and appreciate the content created for it. Notice the hedge here? No repository has unlimited resources (funds, technology, expertise) and therefore it is a given that every repository's provisions for use cases that meet specific goals will be situational, that is, contingent and selective. The challenge for the institutional turn from ingest and storage to access and use will be to understand the difference between scalable core workflows that apply to collections as a whole and special scenarios that require specific tools and services evaluated on a case-by-case basis. I hope that a good understanding of the difference between "core" and "special" will gradually emerge from a conversation among collection curators, digital preservation teams, and interested users. The participation of each of these parties - one of the signal characteristics of the Library of Congress's preserving.exe meeting - will be essential as digital repositories work with researchers to understand software as technology, text and system in all of its conditions and contexts.

An Executable Past: The Case for a National Software Registry

MATTHEW KIRSCHENBAUM, ASSOCIATE PROFESSOR, DEPARTMENT OF ENGLISH, ASSOCIATE DIRECTOR MARYLAND INSTITUTE FOR TECHNOLOGY IN THE HUMANITES, UNIVERSITY OF MARYLAND

> *You are standing at the end of a road before a small brick building.*
>
> Stark words flashed across the network's broadcast channel, like that annual decree going out from Caesar Augustus. Like the first four measures of "Auld Lang Syne." Like the face of a friend bobbing out from a crowd just clearing International Customs, lit in familiarity's halo.
>
> [. . .]
>
> *Around you is a forest. A small stream flows out of the building and down a gully.*
>
> The words filled Jackdaw with a great sense of well-being. Happiness flowed in its own small stream out of Jackdaw's chest and down into his typing digits.

These sentences are from Richard Powers' novel *Plowing the Dark* (2000).[1] The scene, which comes early in the book, has one of its protagonists, a computer programmer and game designer named Jackdaw, sitting in front of his networked terminal during a late-night work session. Someone else posts the opening lines of the text adventure *Colossal Cave*, and dozens of the system's denizens begin a call and response real-time recreation of a game they have all played, using words they all know to describe a place they have all visited, a fantastic place that was formative in their decision to become what they now are.

Colossal Cave Adventure was the original "text adventure" or "interactive fiction," programmed by Will Crowther in FORTRAN on a PDP-10 at BBN in 1975, and expanded a year or so later by Don Woods at Stanford's AI Lab. Thereafter it was ported across almost every platform and operating system imaginable. (I first played *Adventure* in the early eighties on my family's Apple IIe.) As the game's text scrolls by, Jackdaw reminisces about his own childhood encounter with it over a Tymeshare 300-baud modem. And the emotions ring true: *Colossal Cave* was the work that exposed countless young programmers to the possibilities of creating in virtual space, its puzzles and famous "twisty little passages" elevating the experience of coding and debugging to world-making.

Powers' testament to the role of software in our collective memory is no less compelling for its occurring amid a work of fiction. Indeed, consider this: one of our foremost living novelists can knowingly make reference to a twenty-five year-old computer program and be able to count on his audience's powers of identification, just as he would for a mention of a famous painting, film, song, or book. Today you can play *Adventure* on the Web, and you can play it on your

[1] See chapter 16 of the novel.

phone. But while *Adventure* itself is widely available, some of the most basic facts about its creation and history have proven surprisingly elusive. Dennis Jerz, who has done the definitive scholarly detective work on the game's history, notes the widely varying dates that authors and journalists had previously given for its creation, the range spanning the better part of a decade.[2] (That a book or film of any significant cultural import from the latter half of the twentieth century would routinely have its release date misrepresented by such a margin seems inconceivable.) Likewise, the source code for Crowther's original version of the game—before the modifications from Woods—was only recently recovered by Jerz, through a chain of improbable but fortuitous events involving discovery of backup tapes of Woods' original SAIL account, which contained as yet unmodified source files for the game sent by Crowther. This feat was the equivalent of locating an author's manuscript or a film negative, and analysis of the source code significantly revised our understanding of the game, particularly Crowther's original intentions for it—more ambitious then he was typically given credit for. Unlike a literary manuscript or a film master, however, there was no culturally sanctioned depository or repository, no library or archives to bequeath with the newly recovered source code.[3] Like *Adventure* itself, it now simply circulates on the open internet.

Despite the obviously deep emotional resonance we are therefore capable of attaching to computer software, we have generally yet to commit to the scholarly standards, the public infrastructure and institutions, and the individual habits of mind necessary to document its creation and impact as completely and accurately as we do other kinds of cultural artifacts. Here the most basic facts about a computer program that influenced an entire generation might have remained obscure but for the efforts of a single dedicated scholar. And while games might seem like the most replete conduits for channeling such depths of feeling, even seemingly mundane applications, designed for home or office productivity, can elicit strong emotional responses. Precisely because it is always evolving and always obsolescing, we associate specific software packages with different eras in our lives. Nostalgia for *WordStar* or *WordPerfect* isn't just nostalgia for the program it's also a touchstone (or synecdoche) for the 1980s, for MTV and the *Challenger* and Reagan and Gorbachev and all the rest. Perhaps you were sitting in your office immersed in some task in *WordPerfect* or *Lotus 1-2-3* when you heard the Space Shuttle exploded one winter morning in 1986, maybe just as your thoughts were turning towards lunch—that scenario is surely true for many thousands of people.

But there's considerably more at stake than just personal nostalgia. New software is not created *ex nihilo*. The software of today has both an artistic and a functional genealogy to concepts, prototypes, and programs that are now many decades old (Douglas Engblebart's 1968 NLS demo, which showcased the mouse, the graphical user interface, and hyperlinks is often invoked in this context). Yet beyond a handful of well-known exemplars—Englebart's NLS, Vannevar Bush's Memex, Alan Kay's Dynabook, Ted Nelson's Xanadu—much of this pioneering work is increasingly difficult to recapture. Lev Manovich, who is generally credited with having inaugurated the academic field of "software studies," recently published a book about the early history of multimedia. That project, which is exactly what we should expect from a scholar of the computer age, presented considerable practical difficulties at the level of locating primary sources:

---

[2] See Jerz, "Somewhere Nearby is Colossal Cave: Examining Will Crowther's Original "Adventure" in Code and in Kentucky," *Digital Humanities Quarterly* 1.2 (2007): http://www.digitalhumanities.org/dhq/vol/001/2/000009/000009.html.

[3] However the source files for Crowther's original version of *Adventure* were included in the materials archived by the Preserving Virtual Worlds project in institutional repositories at the University of Illinois and Stanford. See PVW's final report for more detail: https://www.ideals.illinois.edu/handle/2142/17097.

> While I was doing this research, I was shocked to realize how little visual documentation of the key systems and software (Sketchpad, Xerox Parc's Alto, first paint programs from late 1960s and 1970s) exists. We have original articles published about these systems with small black-and-white illustrations, and just a few low resolution film clips. And nothing else. None of the historically important systems exist in emulation, so you can't get a feeling of what it was like to use them.
>
> This situation is quite different with other media technologies. You can go to a film museum and experience the real Panoroma from early 1840s, camera obscura, or another pre-cinematic technology. Painters today use the same "new media" as Impressionists in the 1870s—paints in tubes. With computer systems, most of the ideas behind contemporary media software come directly from the 1960s and 1970s—but the original systems are not accessible. Given the number of artists and programmers working today in "software art" and "creative coding," it should be possible to create emulations of at least a few most fundamental early systems. It's good to take care of your parents![4]

Just as early filmmakers couldn't have predicted the level of ongoing interest in their work over a hundred years later, who can say what future generations will find important to know and preserve about the early history of software? (Jerz has established through interviews that Crowther originally designed *Adventure* thinking it was just a game for his children and a few friends to enjoy.)

It is certainly plausible, indeed inevitable, that future scholars like Manovich and Jerz will want to continue to research software for the sake of its history; but, as Powers' novelistic example makes clear, academics, journalists, and the public at large are going to need to know specifics about long-antiquated software as they pursue inquiries involving persons and events from any number of domains—literature, history, the visual and performing arts, government, business, science, medicine, anthropology, musicology, media—from any time from the latter half of the twentieth century forward. Software, after all, is more than just a tool or even a technology; different software packages create the conditions for productivity and creativity in all these many different domains through their individual capabilities, limitations, engineering, and design. Finally, the software industry itself has perhaps the greatest stake in securing the history of its own past practices, both as a source of ongoing inspiration and as a record of its impact on human society. At what point then does responsibility for ensuring the archival basis for such a history pass from individual scholars, collectors, and enthusiasts—or corporate archives which are generally closed to outsiders—to our public institutions and the national public interest?

* * *

"Software," the Yale computer science professor David Gelernter has said, "is stuff unlike any other."[5] But what is software then, really? Is it the source code or the compiled binary? (Of course the former is often a legally protected trade secret.) Is it the underlying math and algorithms, or is it the high-level languages in which these are expressed? Is it just the code or is it also the shrink-wrapped artifact, complete with artwork, documentation, and "feelies," that is extras like faux maps or letters that would become part of the play of a game (see figure 1)? Is it the program or the operating system? What about the hardware? The firmware? What about controllers and other peripherals integral to the experience of a given piece of software? How to handle all the different versions and iterations of software (*Adventure* exists in many dozens)? What about fan-generated add-ons like mods and

[4] The book is Manovich's *Software Takes Command*, Bloomsbury Academic, 2013. See http://rhizome.org/editorial/2013/jul/10/lev-manovich-interview/ for the interview.

[5] In Gelernter's *Machine Beauty: Elegance and the Heart of Technology*, Basic Books, 1998, p. 22.

macros? What about discussion boards and strategy guides and blogs and cheat sheets, all of which capture the lively user communities around software? What about industrial and enterprise software? What about the software in your car or on a 777? What about software that comes to you nowadays as a service, like YouTube and Twitter? What about mobile apps?

Figure 1. "Feelies" that came with John McDaid's *Uncle Buddy's Phantom Funouse* (Eastgate 1992), an early example of electronic literature. They are integral to the experience of the work. Source: http://dtc-wsuv.org/wp/pathfinders/files/2013/08/ubpf-box.jpg

Underlying all of these questions is the more fundamental one of what it means to think of software as a human artifact, a *made thing*, tangible and present for all of its supposed virtual ineffability. Scott Rosenberg, who has written a masterful firsthand account of an ultimately failed software development project, puts it this way: "Bridges are, with skyscrapers and dams and similar monumental structures, the visual representation of our technical mastery over the physical universe. In the past half century software has emerged as an invisible yet pervasive counterpart to such world-shaping human artifacts."[6] Similarly, computer historian Martin Campbell-Kelly has observed that software is characterized by both its invisibility and its universality.[7] On the one hand, seemingly innumerable daily actions are now governed by software, whose variety far exceeds that of mere desktop applications (your phone, your car, the traffic lights you stop your car at, the keycard you swipe to enter your workplace; even your coffeemaker may contain software). On the other hand, there is no way to *see* software—most of us just interact with a user interface or, for the developer, the high-level linguistic abstraction dubbed "source code." But this does not mean that a given piece of software does not have a history, nor does it mean that it's immaterial, that is bereft of the distinctive qualities we associate with more tangible artifacts.

The history and materiality of software can be tracked in numerous ways: its representation in popular culture or a novel like Powers' for example, or else in films. The movie adaptations of Stieg Larsson's Millennium series are notable for their realistic depictions of the software tools the characters use on their computers, while many Hollywood blockbusters give us fanciful and extravagant renditions of software that bear little resemblance to real-world programs. That there are different approaches to representing software in filmmaking speaks to its status as a cultural artifact. Shifting legal opinions around software—its status as expression, object, and commodity—offer another critical window onto societal understandings of what software

[6] In *Dreaming in Code*, Crown Publishers, 2007, p. 8. The failed project was "Chandler," conceived by Mitch Kapor as a successor to Lotus Notes.
[7] See *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*, MIT Press, 2003.

is (or should be). Campbell-Kelly singles out 1969, when IBM was threatened with an antitrust action, as the moment in the US industry when software became commercially distinguishable from hardware, through what was known as the "unbundling" decision. A similar antitrust suit would later arise around Microsoft's bundling of Internet Explorer with its Windows operating system; this time the distinction was between the desktop and the Web. Sometimes software even embeds its own history: Microsoft Word 2.0, released in 1991 at the height of the so-called "word wars" between rival word processing applications, contains an Easter egg (a hidden program feature) that displays an animation of Microsoft coders slaying the "WordPerfect monster" (figure 2). Though seemingly a throwaway gesture, this hidden feature speaks directly to the material element in software, that is the passions, high stakes, and ultimately the humanity (and even humor) that informs it.

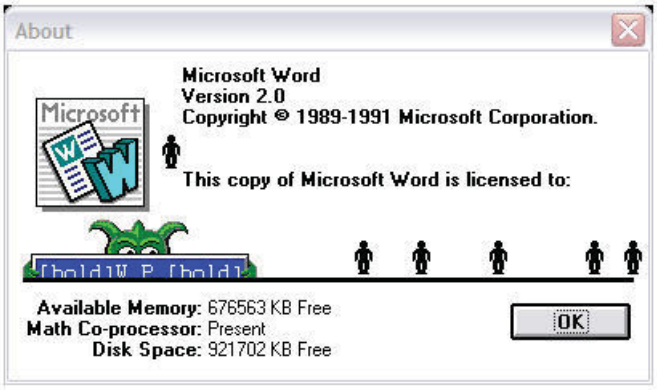


Figure 2. Easter egg activated by a Macro in Word 2.0, illustrating the rivalry with the "W.P." monster. Source: http://www.eeggs.com/images/items/950.full.jpg

Such Easter eggs are not uncommon; indeed, one of the most famous is to be found in Warren Robinett's graphical adaptation of Crowther and Woods' *Adventure* for the popular Atari 2600 platform (1979). As others have argued, this was an important and innovative game, establishing many of the conventions we use to depict virtual space today (such as exiting a "room" through one side of the screen, and emerging in a logically adjoining room on a new screen).[8] At the time Atari's programmers were not credited in the documentation for any of the games they worked on, so Robinett created an Easter egg that allowed a player to display his name on the screen by finding and transporting an all but invisible one-pixel object (see figure 3).

[8] See Nick Montfort and Ian Bogost, *Racing the Beam: The Atari Video Computer System*, MIT Press, 2009 for a thorough discussion of Atari Adventure's technical and creative innovations.

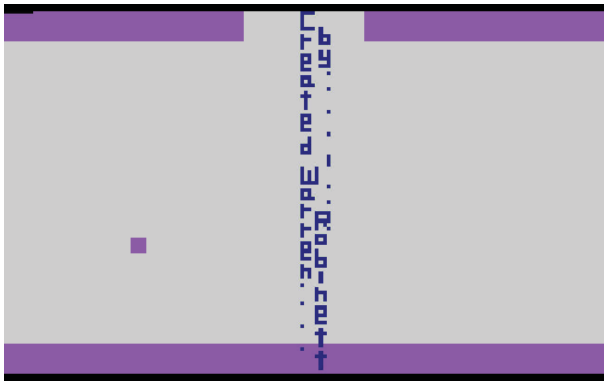


Figure 3. Programmer Warren Robinett's Easter egg in *Adventure* for the Atari 2600. Source: http://en.wikipedia.org/wiki/File:Adventure_Easteregg.PNG

Word of such features would circulate as folklore in the player communities and be passed along in newsletters or (later) on computer bulletin boards. Again, this seemingly slight gesture in fact speaks volumes about shifting attitudes towards software as a cultural artifact. Does "code" have authors? Is software "written" the way we write a book? Nowadays, of course, it is commonplace for programmers to be acknowledged and credited in the software they write, but at the time it required a surreptitious intervention.

We tend to conceptualize software and communicate about it using very tangible metaphors. "Let's fork that build." "Do you have the patch?" "We can use these libraries." "What's the code base?" Software may be stuff unlike any other, but it is still a *thing*, indisputably *there* as a logical, spatial, and imaginative artifact, subject to craft and technique, to error and human foible. Ellen Ullman, who has produced one of our indispensable non-fiction accounts of software development as well as a novel entitled *The Bug*, makes this very point in an interview:

> We have always wanted to rationalize the process of writing software, to have it share the great advances in process engineering that have taken place in manufacturing. We like to talk about software as if it were hardware: we call it a "build"; we talk about "components" and "assemblies." But there is a real heart of chaos in all this. Human beings just do not think and operate like machines, and the ways of human knowledge and understanding do not translate easily or quickly into computer code. That's why most programmers are so wired: there's something obsessional about having to translate the rush of thought into line-by-line statements.[9]

The notion of translating a "rush of thought" into logical, linear statements captures the essence of software development. The languages that software is written in are, after all, *active* languages. They are languages that are meant to be *executed*: compiled, run, and resulting in an outcome that captures the programmer's vision. (The key distinction is between source code, typically written in a so-called "high level" language like BASIC or Java, and object code, which is the source "translated" to the arithmetically executable 1s and 0s of the computer.) To Campbell-Kelly's core qualities of universality and invisibility then, we must also add *executability*. A "piece" of software is not an inert or a passive artifact. It is an active and dynamic one, a set of notations and instructions more akin to a musical score than to a painting or even a literary text. Thomas Scoville gives us a striking account of a programmer who conceives of his coding as akin to conducting a jazz ensemble:

[9] Interview with Ellen Ullman, "Of Machines, Methods and Madness," *IEEE Software* (May/June 1998). The non-fiction account is *Close to the Machine: Technophilia and its Discontents*, City Lights, 1997. It is based on her first-hand experiences working as a software developer in San Francisco.

> Steve had started by thumping down the cursor in the editor and riffing. It was all jazz to him. He could feel the machine sing through his fingers. . . . The first day he built a rudimentary device-driver; it was kind of like a back-beat. Day two he built some routines for data storage and abstraction; that was the bass line. Today he had rounded out the rhythm section with a compact, packet-based communication protocol handler. Now he was finishing up with the cool, wailing harmonies of the top-level control loops.[10]

Such lyrical accounts of the programmer's craft are legion, and could be easily multiplied. What they all have in common is testimony to the fact that writing a computer program is not an abstract logical exercise; it is art and design, tradition and individual talent, and over time the program takes shape as a wrought object, a made thing that represents one single realization of concepts and ideas that could have been expressed and instantiated in any number of other renderings. Software is thus best understood as an artifact: not some abstract ephemeral essence, not even just as lines of written instructions or code, but as something that builds up layers of tangible history through the years, something that contains stories and sub-plots and dramatis personae. Programmers even have a name for the way in which software tends to accrete as layers of sedimentary history, fossils and relics of past versions and developmental dead-ends: *cruft*, a word every bit as textured as crust or dust and others which refer to physical rinds and remainders. (Interestingly, the term has migrated from the name of a physical place, Cruft Hall at Harvard University, to a name for the discarded machinery and hardware components that were visible through the building's windows to its current usage in software development to describe left over code.)

Knowledge of the human past turns up in all kinds of unexpected places. Scholars of the analog world have long known this (writing, after all, began as a form of accounting—would the Sumerian scribes who incised cuneiform into wet clay have thought their angular scratchings would have been of interest to a future age)? Software is no less expressive of the environment around it than any object of material culture, no different in this way from the shards collected and celebrated by anthropologist James Deetz in his seminal study of the materiality of everyday life, *In Small Things Forgotten*. In the end one preserves software not because its value to the future is obvious, but because its value cannot be known. If software is the material manifestation of the universal machine that is the modern computer, then it ought, in theory, to have universal—and unlimited—significance for our understanding of our own past; a past that is not just "usable," as the saying goes, but also, now literally executable.

* * *

The phrase "usable past" comes from a 1918 essay by the literary critic Van Wyck Brooks, published as the United States was on the threshold of assuming its role on the world stage.[11] A relic of an earnest cultural nationalism that now seems crude , it nonetheless spoke to a desire to scour the country's creative heritage for harbingers of a distinct American identity. Institutions like the Library of Congress, the National Archives, and the Smithsonian have all acted upon this vision in various ways, adapting and expanding as new media and new modes of expression take hold in the public life of the nation.

Manovich's comparisons to film history and the visual arts are thus effective in underscoring just how impoverished our documentary efforts for software have been to date. Yet film preservation itself is a notoriously volatile enterprise and would not always have seemed a promising exemplar. By many estimates over 80% of the American films produced before

[10] In Thomas Scoville, *Silicon Follies: A Dot. Comedy*, Atria Books, 2001, p. 72.
[11] The full title of the essay is "On Creating a Usable Past." It was first published in the *Dial* 64 (April 11, 1918).

1930 are now lost—casualties of their volatile nitrate base. Today the Library of Congress's Packard Center for Audio-Visual Conservation (also known as the Culpeper facility) is charged with the long-term safekeeping of the nation's films, television programs, radio broadcasts, and sound recordings—the largest audio and moving image collection in the world. For decades the Library of Congress has also been receiving computer games, and in 2006 the games became part of the collections at the Culpeper campus. But while the Library registers the copyrights, what it means to preserve and restore vintage computer games—or any kind of computer software—is less clear. As yet there is only the beginning of a national agenda for software preservation, and precious little in the way of public awareness of the issue.[12]

In 1988 Congressional legislation established the National Film Registry, whose mission is to preserve films deemed "culturally, historically, or aesthetically significant."[13] The timing was not accidental. The colorization of classic films was becoming more commonplace, notably as a result of Ted Turner's acquisition of MGM's catalog. For those titles entered in the Registry, the legislation prohibited the distribution and public showing of any film colorized or otherwise edited without explicit labeling indicating exactly what had been changed. Thus a key mandate of the law was to ensure the fidelity of these films to their original versions, even as technology allowed for ever more possibilities for enhancement.[14]

Should there be a National Software Registry (NSR) modeled on the same basic principles and intentions? This suggestion was floated by Clifford Lynch and others at the Library of Congress's May 2013 Preserving.exe meeting.[15] This essay argues that such suggestions should be taken seriously. Like film, there is a general sense that an important class of cultural artifact and expression is now imperiled. While software preservation may not necessarily present any one single catalyst like colorization as the impetus for the creation of such a registry, it is widely recognized that many different factors, from the degradation of physical media to the obsolescence of the hardware and operating systems that create software's dependencies, are lending the issue urgency. Some might also wish to argue that while films are vehicles of cultural and imaginative expression—stories and artworks that tell us about ourselves—software is strictly utilitarian, a functional often commercial product that exists for getting work done. Of course such utilitarian definitions immediately miss a whole range of important software genres, notably games but also other forms of interactive storytelling and digital art. This essay, however, has also tried to suggest why even so-called utilitarian software benefits from being regarded as a cultural artifact and expression, a product of a particular place and time with its own inherent stories and histories. And while it is true that software is not nearly as neatly defined as film, it is important to remember that many of the films now part of the Registry originated as other than traditional narratives. Also included are early silent documentaries, slice-of-life and incidental footage; as well as home movies (Dave Tatsuno's *Topaz* and the Zapruder film), civil defense films, animation, experimental film, and even music video (Michael Jackson's "Thriller").[16] Software too has diverse genres,

[12] For example, one might note that there is no mention of software or executable content in the NDSA's 2014 National Agenda for Digital Stewardship: http://www.digitalpreservation.gov/ndsa/nationalagenda/.
[13] H.R.4867, Public Law No: 100-446. The full text of the legislation is available here: http://thomas.loc.gov/cgi-bin/bdquery/z?d100:HR04867:@@@L&summ2=m&
[14] The National Film Registry's online presence is here: http://www.loc.gov/film/.
[15] The full title of the meeting was Towards a National Strategy for Preserving Software. See http://www.digitalpreservation.gov/meetings/preservingsoftware2013.html the meeting's agenda, participants, and selected presentations and reporting.
[16] The 2011 documentary *These Amazing Shadows* by Paul Mariano and Kurt Norton is an excellent introduction to the history and contents of the National Film Registry.

ranging from games to office productivity tools to utilities, middleware, and many others. It cannot be essentialized as any one thing.

Titles are entered in the National Film Registry via a public nomination process and the subsequent deliberations of the Library of Congress's National Film Preservation Board. Films must be at least ten years old to be eligible. While active measures are then taken to preserve (and if necessary restore) the films thus selected, these tangible outcomes arguably are subordinate to the simple power of the Registry to create public interest and awareness. Lists, after all, are powerful focalizers for our collective attention. Lists of great films, great books, and great albums are commonplace; even before the internet made list-making (and critiquing) a popular pastime for many, publications such as the *Book of Lists* highlighted the importance of lists as ways to consolidate opinion and knowledge. Thus the mere process of amassing lists of software suitable for inclusion in a National Software Registry would serve to focus public attention on the subject.[17] Such lists have been made before. For example, in 2006 Henry Lowood, Warren Spector, Steve Meretzky, Matteo Bittanti, and Christopher Grant compiled a "Game Canon," a list of 10 computer games submitted to the Library of Congress for preservation due to their cultural significance.[18] While the choices were well-considered and well-argued, the list, predictably, was highly contentious. This is as it should be, and underscores the importance of balancing public opinion and expert consensus in any decision making process. Lists tell us as much about the people who made them as they do the items that inhabit them. The value of the selection process should not be underestimated, quite apart from whatever practical measures are taken to preserve the software titles thus selected: therefore the question of the creation of a National Software Registry should not be dependent on the current state of the art in software preservation or the projected feasibility of "preserving" any one specific title.

While the particulars of an National Software Registry's governance and operating principles—such as how many titles per year or whether there is a "waiting period" akin to the ten years for the National Film Registry—are best left to others, as are considerations related to its technical infrastructure and implementation, here are some guiding principles I would put forward to help focus the issues where consensus would need to exist:

- An NSR should have as its most important criteria software that has had a significant cultural, historical, aesthetic, or empowering impact on the public. Both technical innovation and degree of popular uptake should be regarded as relevant to assessing these criteria, and may be evaluated independently.
- An NSR should have as its mission the documentation, preservation, and accessibility of such software. Additionally, an NSR should act to raise public awareness of the importance of software preservation and the role of software in the life of the nation.
- An NSR should be multi-lingual and represent the diverseness of the national experience.

[17] In July 2013 the author was asked to compile a list of the "10 Most Influential Software Titles Ever" for *Slate* magazine. That list is available here: http://www.slate.com/blogs/browbeat/2013/07/30/_10_most_influential_software_programs_of_all_time_from_sabre_to_minecraft.html. Far more useful and revelatory, however, are the hundreds of comments the piece attracted, many of which offer well-considered arguments for specific titles omitted, as well as more general selection criteria.

[18] See http://en.wikipedia.org/wiki/Game_canon.

- An NSR should be sensitive to the acute limitations of a strictly nationally-scoped collections policy .
- An NSR should have an open and public nomination and selection process, and solicit participation from experts in industry, academia, journalism, and the public at large.
- An NSR should be capable of acting as a trusted repository for verified authentic object code and insist on the availability of executable object code as a pre-condition for inclusion.
- An NSR should be capable of acting as a repository for verified authentic source code, and advocate for the inclusion of source code in the strongest possible terms; nonetheless, an NSR *should not insist* on the availability of source code as a pre-condition for a title's inclusion.
- An NSR should advocate for appropriate licensing and where necessary appropriate exceptions to such legislation as the DMCA to provide for the archival documentation and preservation of software.
- An NSR should cultivate industry partners to raise awareness and create support for its mission.
- An NSR should cultivate partners in the open source community to raise awareness and create support for its mission.
- An NSR should be agnostic with respect to free, commercial, or proprietary software.
- An NSR should be agnostic with respect to high-level languages, platforms, operating systems, hardware, and storage media.
- An NSR should conceive of its mission in terms of access whenever possible, and work with academic, industry, and public partners to provide solutions for the display and execution of legacy software.
- An NSR should act to enable research in best practices for software documentation and preservation.
- An NSR should act to document and collect not only software but also the material culture of software, including manuals and documentation, packaging, inserts, "feelies," and other accessories, popular computer magazines and books, images (including screenshots), video (including screencasts), harvested Web content, fan-created content, and development and design documents.
- An NSR should work in cooperation with the National Software Reference Library at NIST[19], the Internet Archive, existing or future museums and collecting institutions, and individual collectors, hobbyists, and fans.

[19] As described on its Web site, the National Software Reference Library "collects the original media for off-the-shelf software. This information is processed to obtain digital signatures (also called hashes) that uniquely identify the files in the software packages. With the signatures, law enforcement investigators can automate the processing of these files on seized computers, system administrators can identify critical system files that may have been perturbed, digital archivists can identify applications versus user-created data, or exact duplicates of files." As of this writing, the NSRL contains some 20,000,000 unique hash values. While obviously a resource of tremendous importance that should have a role in any conception of a National Software Registry, the NSRL is not a public-facing collection, nor is it designed to act as an advocating entity in industry or with the public at large. It has no provisions for the circulation of software, nor does it facilitate the execution of legacy code. Finally, it does not collect manuals or other forms of material documentation. See http://www.nsrl.nist.gov/.

Of these guiding principles the most controversial is likely to be my recommendation with respect to the inclusion of source code. Numerous experts have insisted on the importance of source code for software preservation and understanding our executable past.[20] I too endorse such arguments. However, the reality is that the source code for some programs which would otherwise be obvious candidates for inclusion is going to remain a protected trade secret for the foreseeable future. An NSR that conspicuously excludes historically significant software on the basis of the absence of source code is going to be incomplete, impoverished, and will dilute its relevance and efficacy through these obvious omissions. The reality too is that some source code may simply no longer be available or locatable. But the absence of source code does not mean that there are no preservation and documentation actions that can be undertaken. (There may also be scenarios whereby an NSR could act as a depository for protected source code, to be made public after a specified length of time.) In some circumstances, techniques such as reverse engineering and disassembly may also render the need for source code less acute. It is thus my view that insisting on the provision of source code as an absolute condition for inclusion in an NSR would create insurmountable obstacles, prove an unnecessary distraction, and prohibit positive steps that could be taken to ensure the preservation of even proprietary software. Nonetheless, this is the single most important issue for an NSR to consider in its foundation, and arguments on all sides should be heard and evaluated.[21]

Just as the then still-new medium of film was perceived as disposable or ephemeral or frivolous and forgettable in 1918—in short, hardly the stuff of a usable national past in Brooks' exclusively literary account—so too is software currently in danger of having its heritage eclipsed by the limitations of our own present moment. At the Preserving.exe meeting a representative from Github, the massive open source online repository, made the connection to such high-minded ideals all but explicit, declaring the software culture on the Web a new cultural canon and invoking the likes of Emerson, the Beowulf poet, and Adam Smith. But software is not just another medium or art form. It is in fact a meta-medium, one that has grown to encompass a variety of other cultural functions and expressions, from commerce and industry to art and entertainment—not the least of these being filmmaking, since movies are now routinely created and processed using digital software.

Like film history, much early software history is already at risk, and some of it certainly lost forever. But that is all the more reason for software preservation to take its place as part of the national cultural heritage agenda, and to increase the public's awareness of the importance of software in the life of the nation. The serious consideration of the case for a National Software Registry is a means toward those ends.

[20] See, for example, John G. Zabolitzky, "Preserving Software: Why and How," *Iterations: An Interdisciplinary Journal of Software History* 1 (September 13, 2002): 1-8, and Leonard J. Shustek, "What Should We Collect to Preserve the History of Software?" *IEEE Annals of the History of Computing*, 28.4 (2006): 112–111. Zabolitzky is uncompromising on this point: "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."

[21] An opposing argument might contend that insisting on the availability of source code for inclusion in the NSR would be an incentive that would encourage software manufacturers to become less protective of their legacy code.

We didn't see this coming: Our unexpected roles as software archivists and what we learned at *Preserving.exe*

ALICE ALLEN, PRIMARY EDITOR OF THE ASTROPHYSICS SOURCE CODE LIBRARY AND PETER TEUBEN, ASSOCIATE RESEARCH SCIENTIST, ASTRONOMY DEPARTMENT, UNIVERSITY OF MARYLAND

BEFORE PRESERVING.EXE

The idea that software should be preserved, archived, for its own sake was not really on either of our radars before hearing about the Preserving.exe Summit. Certainly we both have kept old software around for our own personal use (in part to be able to retrieve data) and one of us (Alice) has saved packages of old software at work to eventually present to retiring coworkers ("As a souvenir of your pioneering use of Visicalc, I'm pleased to give you the most recent version available…").

Our involvement in the Astrophysics Source Code Library (ASCL), a free online registry for source codes used in astrophysics research, arises from our interest in making software used in research discoverable in order to increase the transparency of astrophysics research; software used in research is a method, and should be open to examination just as other methods are. We register old and new source codes for astronomy to make the science understandable, to increase its transparency and preserve the integrity of the science. In addition, the science benefits from the reuse and further development of codes; many authors encourage enhancement of their codes by others to better benefit the community.

To be discoverable, astronomy software must of course be available, preserved; rather than storing source code, the ASCL generally points to a code's location, which is usually on a university or personal site or in a social coding repository such as GitHub or SourceForge. We are not aware of any large-scale effort to preserve copies of astronomy software. The Summit convinced us that without such an effort, some of the greatest artifacts of the community's creative problem-solving are at risk of being lost.

OBSERVATIONS

Participant presentations educated us on other archiving efforts and different views on software preservation, and inspired changes to our own project. We discuss three of the overarching ideas and challenges for archivists that the Summit brought out.

Software as a cultural artifact

The Summit broadened our view and appreciation for software as a cultural artifact and as a method of capturing creativity in problem-solving, whether the problem is writing an engaging computer game for entertainment or software to model gravitational effects on colliding galaxies.

Now we see the loss of computational methods that result in research as a loss of part of astronomy's cultural heritage. This isn't happening just for astronomy, of course; the Summit made clear that it is happening for *everything*. With so much rendered digitally, whether born that way or migrated to a digital medium, without preserving the digital artifacts and the software (and sometimes hardware) to lift these artifacts from their digital storage, we risk losing our art, our music, our games, our prose, our data, and our histories, of daily life and activities, of solutions to scientific problems, of popular pastimes and play experiences, and even knowledge of our computer worries and angst such as that (which once loomed so large and now seems so distant and almost quaint) over Y2K, a worldwide issue. Say "Y2K" to a class of high school seniors today, and see how many of them know what you're talking about!

Though we have always considered problem-solving, and therefore authoring software, as a creative endeavor, Rachel Donahue from the University of Maryland's Maryland Institute for Technology in the Humanities, in her talk *Saving the Software Present to Read our Robot Future,* went further and stated that she sees software as an art form, one that can and should be studied. As an archivist, she wants to preserve, and is preserving, not only the finished software, but also the artifacts of its creative path, of its development: its earlier versions, design documents; pseudocode written in margins, and correspondence. (How much correspondence is lost by hitting *delete*!) She pointed out that a student of videogame design may want to study what came before just as visual arts students study the sketches of old masters.

Mozilla's Otto de Voogd and Robert Kaiser's talk *Unlocking the Potential* showed that Mozilla goes even further: Mozilla not only archives all of its source code, releases, and executables, but also its mailing lists, bug tracker and analysis, all discussions and code changes, why the changes were made, and its traceability matrix in development and for software fixes, all of which is available online. In response to pre-workshop questions, Otto mentioned that one of the reasons for preserving software is "the discovery of prior art," which we took to mean art that may not have been recognized as such in its time. In addition to the possibility of such discovery is the need to preserve current art; Mark Mansfield of the Smithsonian American Art Museum curates born-digital time-based media art such as Cloud Music, a mixed media project that turns the movement of clouds into sound. Software enables a new performance space for creativity; such software-dependent art presents unique challenges not just for its exhibition, but also for its preservation.

In a similar vein, Doug Reside of the New York Public Library (NYPL) mentioned in his pre-conference comments that the NYPL is increasingly receiving born-digital content from artists created with proprietary software; copies of this software is necessary to make the content – the art itself – accessible. He also preserves set and lighting designs for theaters and said in his presentation *Serving Born Digital Video* that he is able to recreate these through emulation, allowing users not just access to these designs, but the ability to manipulate them, too.

There was a lot of interest in and attention given to the preservation of video games. Games represent many people's first deliberate use of software, and its archiving comes perhaps more easily than other kinds of software, as it is ubiquitous, inexpensive, and very familiar. In a very exciting break-out discussion, the opinion that archiving games seems useless was

countered by Megan Winget (University of Texas at Austin) and others vigorously defending the practice. Megan pointed out that games use a well-defined set of software and can also serve as an easy measure of success to demonstrate that emulation works: a child playing the game smiles. Certainly games embody a great deal of creativity and art and some become deeply engrained in our culture, with elements from them, especially characters, showing up on television and in movies, as toys, in popular literature and other media.

Determining what to preserve

Those preserving software cannot know what will be needed, useful, or helpful in the future nor how the software and related materials may be used; this was a recurring theme of the Summit. Doug Reside asked, "*What are the things that we can safely say no to archiving?*" Even with our very tight focus on astrophysics codes, we have to make decisions as to what is suitable for our resource. A founding principle of the ASCL was to register software written by scientists used in refereed research; should we also register software used by a telescope to reduce data used in research if the code itself is not mentioned in any journal paper? Should we include software used to control telescopes?

We have looked to other sciences to inform our efforts in astrophysics, to see what is archived, and whether and how it is available. Our finding is that code archiving across scientific disciplines is very uneven. This was corroborated at the Summit: Debbie Douglass of the MIT Museum pointed out that different disciplines have different philosophies about archiving materials. She said that aeroscience, for example, saves everything; doing so is part of the culture. Left to their own devices, those in life sciences will save nothing – they don't have the habit of archiving. She finds software an interesting case and stated there are actually a lot of specialized (sector) software archives.

Other software preservation efforts like the ASCL with very narrow mandates face the same "what to save" issue; Amy Stevenson of Microsoft Archives (*Software Preservation in a Software Company: The Shoemaker's Children...*,) stated she cannot collect everything. She preserves Microsoft software and the artifacts around its finished products: its documentation, books, even the marketing for the software, and what is competitively important or may become competitively important, whether produced by Microsoft or by competitors.

The National Institute for Standards and Technology's National Software Reference Library (NSRL) preserves only executables; it does not collect source code nor the artifacts of its creation. Barbara Guttman reported that this effort started as a way to help the law enforcement community with computer forensics, and that the NSRL has been expanding into software identification to support computer security.

From a breakout discussion, one group pointed out the value of heterogeneity in the archiving community, as the goals for archiving vary and what is saved will reflect this. The organizations represented at the Summit ranged from those with a specific focus to their archives, such as Mozilla, the Microsoft Archives, and the ASCL, to those with far more material to consider for their collections. The Library of Congress, NSRL, university libraries, the Smithsonian museums, the Internet Archive, and others all have much more to consider, a much broader range of software to think about, than those with a more narrow focus, but even so, this underlying issue is the same for all of us: *what are the things that we can safely say no to archiving?*

Running and using preserved software

Preserving software is important, but how useful is it if there is no way to run it? One of our concerns is software rot, the inability to run a code on current hardware or under current software. One code author recently wrote:

> *Certainly, I get about one e-mail a year about the Figaro data reduction code, and it's almost always: "I upgraded to the latest OS X version and now your program doesn't work any more..."* (Keith Shortridge, personal correspondence)

With the accelerating pace of technological change, the inability to run archived software is certain without mitigation strategies against rot. Software that rots may leave behind orphaned data, information to which individuals and organizations need access but can no longer retrieve. The NSRL exists in part to ensure that law enforcement can access information needed for its investigations and, as pointed out the article Life-Saving: The National Software Reference Library (one reading for the Summit), because it had the software to access orphaned information, even had an important role in saving someone's life. Mozilla's Robert Kaiser mentioned the need for standard data formats in the talk *Unlocking the Potential*; he would like to be able to open his own data. In discussion, Robert Hanisch (STScI/ASCL/VAO) stressed that astronomy went to a standard data format 30 years ago, a format still in daily use. Having a standard flexible data interchange has enabled a great deal of research and allowed astronomy to grow as a science.

In his talk *Hardware and Emulation to Access Creative Computing*, Nick Montfort of the Massachusetts Institute of Technology made clear that both running software on original hardware (which has its own complexities) and in emulation have their benefits. A key benefit of running software on its original hardware is to provide a client with an authentic experience in using the software. In an open discussion, the point was made that eventually, even if the hardware remains usable (which is questionable over the long-term), the people who can bootstrap the systems and troubleshoot problems with the hardware will no longer be available.

Henry Lowood of Stanford University Libraries stated that people are working on migrating software and data off the original media and from the original hardware. He sees two eventual options: running the software will be up to the user, or software will run in emulation.

Clifford Lynch (Coalition for Networked Information) said outright in his remarks (*Putting Software Preservation in Its Broader Context*) that the future of preserved software use will be through emulation; the experience of using original hardware will be lost. He tempered this by observing that libraries have 18th century books, but they don't preserve the experience of reading that book in the 18th century, without electric lights and other modernities.

The Olive Archive Project (*Not a Graveyard: The Olive Archive Project*) was presented by Erika Linke and Dan Ryan from Carnegie Mellon University. This project takes the approach of creating virtual machines and running software natively on those machines. Erika and Dan want to preserve not only the code and data, but also the experience of using the software. In a demonstration of the project, a virtual machine was streamed live over the internet that allowed Dan to play Doom in an emulator. Megan Winget stated a need to archive the documentation of the users' experience in running and using software; in later discussion, the question as to how to evaluate the authenticity of the user experience and whether anyone has been researching this was asked. No one had an answer for that.

It certainly makes sense to us, as owners of a mostly working Kaypro-4 (Peter) and Apple II+ (Alice), that emulation will be the way the vast majority of preserved software will be run; old hardware and storage media will not work forever. For our project, running code can be important, and we think the Olive Project approach could be used for emulating astrophysics codes that current operating systems and hardware can no longer run. Running these codes in emulation may have real benefits to the science; for example, codes that had taken six months to run to generate results when the codes were new would run at much greater speeds, allowing multiple iterations of an experiment that the original investigator may have been able to run only once.

Even if codes cannot be run, looking at them has value. In his talk *Reflections on Users and Stewardship of Software*, Josh Greenberg from the Alfred P. Sloan Foundation made the point that seeing what makes software work, understanding the source and what it's doing and how, has value. We agree, and this is why we seek source codes rather than executables. Scientists should be able to see what assumptions, equations, values, and algorithms have been used to generate results; this is one way the transparency and integrity of the science is maintained.

HURDLES, OPPORTUNITIES, AND RECOMMENDATIONS

Small group and open discussions helped participants identify hurdles to and opportunities for preserving software. We address some of them here in no particular order and include some of the relevant recommendations made during the Summit.

Access to software and materials

Acquiring software to archive continues to be an issue; how do we find it, and what happens when we do? We need ways to find and reach out to those who may have old software. If one of the Summit participants comes across materials that are not compatible with his or her project, there should be a way to channel that material to an archival effort that can make use of it. In addition, we have seen codes that had been downloadable from online sources disappear when code authors change institutions, leave the field, or die.

Jason Scott (Internet Archive) offered to pull together a mailing list that those at the Summit could use to circulate information about available materials as an informal way to help channel available materials to those who might be interested in it. The invitation to join the mailing list was sent just a few days after the Summit ended.

Making use of the Internet Archive's Wayback Machine is a useful strategy for software that at one time has been available online. This is something we have seen with astrophysics software; a resource called AstroTips.com, which tracks astronomy software used by amateur astronomers, currently uses links to the Wayback Machine for software not otherwise (still) available online, such as the source code called Yomama. We have not yet started to use the Wayback Machine for this purpose, but will start downloading "lost" software from this valuable resource to preserve it in the ASCL.

Joel Wurl (National Endowment for the Humanities) was struck by the number of comments about storing the ancillary documents and artifacts around the software, not just the software itself, and stated that there are collecting organizations that have some of this other material and they need to know there is a broader objective. Abbey Potter (Library of Congress)

asked whether there are other communities we should do outreach to. Though that didn't get answered directly, Debbie Douglass said we can reach out to other academics, not just historians. She suggested that we have to think about how our archived resources can be used; that may help inform us as to how we can acquire material.

Access to and acquisition of materials may be accomplished by reaching out to collectors; many collections, including the Internet Archive and the Boston Computer Museum (now the Computer History Museum), got started and have the bulk of their material coming from dedicated collectors, from private sources. We can collaborate to help find homes for materials that are offered to us but that we don't need – this was a main reason for Jason to assemble a mailing list. "Well, we don't collect that, but *this person* does," or "Let me see who I can find who may be interested." These informal networks can be very useful in building collections.

The ASCL acquires software primarily by hunting it down; we skim journal articles (that then often refer to other journal articles) to look for codes used in the research, searching for any code mentioned online, and if we do not find it, asking the author of the code for its location or, if none exists, an archive file of the code. Most of our requests to authors go unanswered (67%) or are declined (20%). Our goal is to have a substantial number of code authors and journals request registry in the ASCL, and though we do see movement toward that goal, we suspect it is still several years away.

Copyright and DMCA

Much of the preserved software from the early days of personal computers is available because dedicated hobbyists saved it, sometimes having broken copy protection to do so. The question arose as to what happens when the only copy left is an illegal copy – should it be archived?

The Digital Millennium Copyright Act (DMCA) is seen as an impediment to archiving. In his talk *Optical Media Mass Ingest*, Paul Klamer of the Library of Congress, which has an exemption from DMCA that allows them to break encryption on materials, informed us that of the DVDs that come in for copyright reasons, 20% aren't readable when he tries to take the .iso image for archiving. Nick Montfort stated that though there are significant technical challenges, they are not the bottleneck; the hindrances are copyright. Rachel Donahue noted that intellectual property rights had come up several times in the discussions at the Summit. She pointed out the need for legislative support that allows archiving intellectual property without violating copyright and patent laws.

As commercial software ages and is replaced with newer versions or different products, it loses its value as a commercially viable product; this provides an opportunity for archivists, who can ask for permission to release it. As of this writing, the Computer History Museum (CHM) is offering the source code for Adobe Photoshop 1.0.1 as a download; Adobe has made the code available through the museum for examination and study. As the CHM states, "*Software source code is the literature of computer scientists, and it deserves to be studied and appreciated.*" This echoes back to Rachel Donahue's statement on the value of software as an artwork worthy of study.

Closed/walled systems

Software that is walled off, such as Facebook, *The Washington Post*, and anything in the cloud cannot be archived, nor can it be emulated well; it would be an astronomical effort to

recreate the function or experience of using these systems without access to the software and other key elements such as highly time-dependent databases.

In thinking this over, we came to realize there are ways to demonstrate or imitate the user experience. Just as for the 400th anniversary of the construction of Galileo's telescope people could order kits for putting together a plastic replica of Galileo's telescope and on Sunday evenings, we can listen to 1950s radio shows on National Public Radio complete with Maxwell House ads, we might encourage the screencasting of software sessions so future archivists can show patrons what using today's software looked like. Though screencasting could preserve what these systems look like, unless the proprietary software is being archived by the companies producing it, this software may eventually be lost.

That said, Ben Balter of GitHub pointed out in his presentation *The Next Cultural Commons* that 75% of the internet runs on open source software, and increasingly, open source software is used by governments, corporations, and individuals. The hope that GitHub, other collaborative software sites, and open source software holds out for software in general does not apply to closed systems. The issue of preserving software living in a walled garden remains for future discussion.

And the rest (the Professor and Mary Ann)

We cannot fully cover the information exchanged in this two-day meeting in this brief report, but amongst the many other things discussed, we want to mention briefly these few additional items.

> *People producing born-digital works don't know to preserve them nor how to preserve them, and start-ups don't know the importance of what they are doing and don't archive as they go.*
>
> The necessity to archive goes back to the point that we don't necessarily know what will be useful in the future. Recommendations for dealing with these issues included asking for advice from experienced archivists to help us learn what might be useful and should be archived, raising awareness of the importance of archiving, and for software developers, providing training such as that offered by Software Carpentry and guidance on coding practices such as versioning. Making the case for preserving software by showing how preserved software has been used may also be useful.
>
> *User expectations vary across constituencies.*
>
> What, how, and why we collect influences who our users are and what their expectations for our collections are. The archival requirements for different types of artifacts and offerings vary and also depend on what our collections are trying to do and how they are used. We could perhaps collaborate to come up with lists of what needs to be considered for different types of software and other materials. This was framed as a list that answers the question "What do I need to think about now that *this thing* has been offered to my collection?" Such checklists would help people consider all angles of an acquisition.
>
> *We need funding and resources to do what we do.*
>
> This is an issue we struggle with mightily, as all of our ASCL efforts are volunteer! As our effort evolves, some of its future possibilities depend much on funding and community input. So it is with other efforts represented at the Summit. We (all of us)

must build our cases for the importance of preserving software and reach out to funding organizations to share and impress upon them the value of what we do.

As software is lost, there is a loss of human knowledge over time.

This is a two-fold problem; the loss of software and its artifacts is one type of loss, and the loss of people familiar with the software and hardware, who know how it should run and how to make it run and to repair or rebuild it, is another. Recommendations to mitigate these losses included archiving anything and everything, not just software, but related artifacts as well, increasing the visibility of the need for archiving, and using of original hardware where possible and using emulation to recreate the experience of software use when use of original hardware is not possible or desirable. Greater visibility of archiving efforts and greater coordination among archivists were also mentioned.

This argument for archiving – the loss of human knowledge and of astrophysics culture – had not occurred to us in quite those terms. We have made the philosophical arguments regarding science transparency and reproducibility for preserving astrophysics codes, but not the cultural arguments, and now consider doing so.

TOWARD A NATIONAL STRATEGY

A national strategy will be challenging to develop and implement. We envision difficulty in meeting the needs of the different interest groups and fields. However, we feel one step in the process is to require public release of software developed through government-funded research (absent compelling reasons, such as national security). Further, efforts should be undertaken to ensure release actually takes place. Currently, government policies regarding these products of research are uneven, with some funding programs requiring release of research artifacts such as codes and data and others not. As such products are taxpayer funded, these policies should be uniform and strongly in favor of public release. Making the software available not only serves the immediate public interest, but also makes the software available for archiving.

Conclusion

What did we get out of these discussions? We are very impressed at the lengths to which others go to preserve so much surrounding software and its development and use; we have not been preserving astrophysics source codes, simply serving as a registry of them as so many scientists prefer to keep their codes close to them (for a variety of reasons, some of which they may have no control over, such as intellectual property issues). Clearly there are lessons to be learned from others who preserve software, but as was pointed out in the meeting, there are also lessons to be gleaned from previous disparate archiving efforts.

As a result of this Summit, we have now started to archive at least one version of each code where we can and as time allows, not to serve it to the public, but simply to have it archived in case the code disappears into the black hole of deleted webspace. The idea that *we don't know what may be useful in the future* struck us deeply, and as we wrote this report, one of us (Peter) often stated that phrase as we debated what to include. Because we don't know what (else) you, our reader, may find useful, we have posted the rough notes we took during the meeting online.

We are working to preserve the integrity of a science through making codes discoverable for examination, and so have become accidental software archivists; to know that others are wrestling with many of our same issues (though on a much larger scale), and that we might find not only common ground but also common solutions, is enlightening and comforting.

Life-Saving: The National Software Reference Library

INTERVIEW WITH DOUG WHITE, PROJECT LEADER FOR THE NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY NATIONAL SOFTWARE REFERENCE LIBRARY BY TREVOR OWENS, LIBRARY OF CONGRESS, MAY 4, 2012

Insights is an occasional series of posts in which members of National Digital Stewardship Alliance Innovation Working Group take a bit of time to chat with people doing novel, exciting and innovative work in and around digital preservation and stewardship. In this post, I am thrilled to have a chance to hear from Doug White, Project leader for the National Institute of Standards and Technology National Software Reference Library. I heard Doug give a fantastic talk about his work at the CurateGear Workshop (see slides from the talk here).

Trevor: Before we dig into the details of the project, you mentioned that the NSRL has already resulted in saving at least one person's life. Could you walk us through exactly how that came about? I think it makes for a really compelling story for why software preservation matters.

Doug: Certainly; it was an unintentional circumstance. To begin, we often were asked if software may be borrowed from the NSRL, and the response was, "No, we are a reference, not a lending library." But then we received a call from an Food and Drug Administration agent on a Friday afternoon in December 2004.

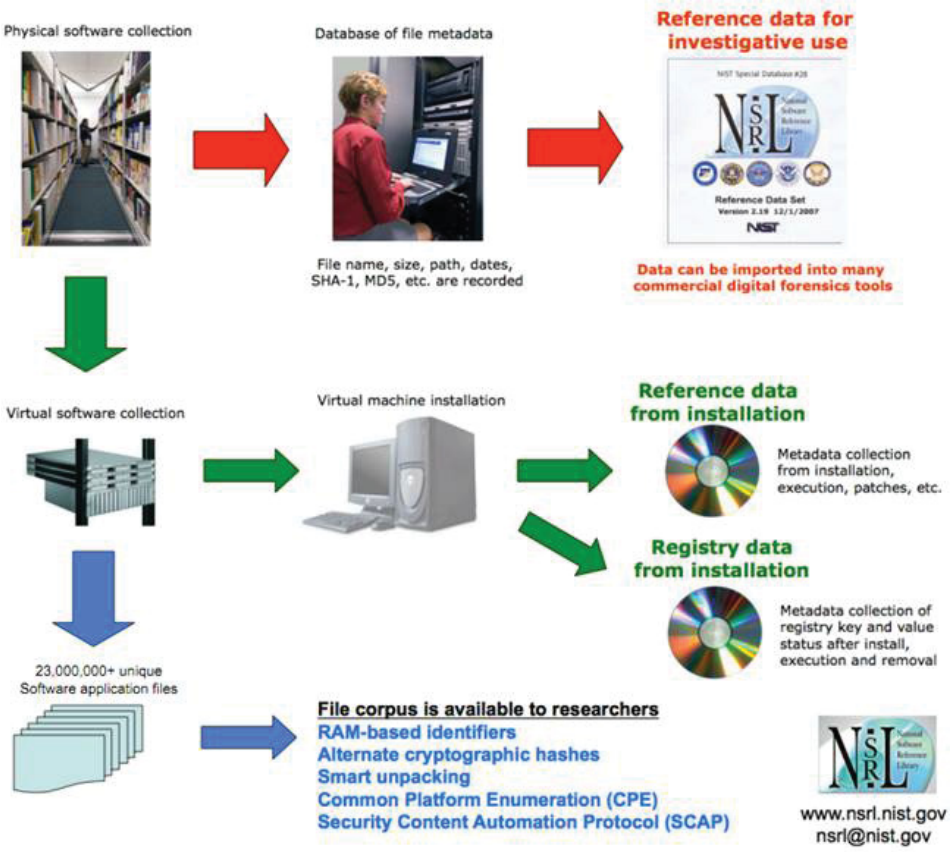
A medical supply company in Miami had received a delivery of botulin, which was to be processed into Botox and distributed. However, it was misprocessed, and a dangerous concentrate was distributed. The FDA had all of the information needed to identify the recipients, but the information was in a file created with a 2003 version of a popular business software application. The 2004 version available to the FDA could not open the data file. The manufacturer of the software was also unable to supply the relevant version.

It so happened that one of the agents involved in the case was familiar with the NSRL, and had in fact provided software to us earlier in the year. He called, explained the situation, and asked if we had the 2003 version of the software. We did! The agent then arranged for an FDA contact to come to NIST, get the software, and put it on a jet to Miami. The people working the case in Miami were able to install the old version, open the data file, and trace the paths of the botulin.

Several fortunate events occurred to enable this story to end on a positive note. We have a process in place should this occur again, though we consider the NSRL to be a "last resource."

Trevor: I have heard you describe the National Software Reference Library as a library of software, a database of metadata, a NIST publication and a research environment. Could you give us a little background on the project and explain how NSRL serves these different functions?

Doug: The diagram below is an overview showing several facets of the NSRL. The path using red arrows involves our core operations, green arrows designate "derivative" operations, and blue illustrates some collaborative research.



The physical library is our foundation. At the inception of the project, in 2000, organizations were creating and sharing metadata describing computer files on a very ad hoc basis. If the metadata were questioned, it was highly unlikely that the original media were available to resolve the issue. The NSRL operates in the same fashion as an evidentiary locker, with the original media available in the event of a question.

The physical library has a parallel virtual library. NSRL has created bit-for-bit copies of the original media and images of packaging materials that are kept on a network storage device. I need to point out that the NSRL runs on a network disconnected from the Internet, and in fact, also disconnected from the NIST network infrastructure, using equipment and cables we installed. The media copies can be manipulated automatically, used by multiple processes and repeated physical contact with original objects is minimized.

From the packaging and media, we collect metadata from every application, from every file. We store the metadata in a PostgreSQL database. The database has several schemas, which act as conceptual boundaries around accession processes, the collection of software application descriptions by manual processes, the collection of content metadata by automated processes, storage processes and publication processes. The work processes and the technology are modular components that are easy to test, maintain, train, or reuse. The database metadata (with the exception of staff information) is available on request.

There is a subset of the collected metadata which is of use to investigators and researchers in the community in which NSRL participates, and the subset is published quarterly as NIST Special Database #28. The specific data includes:

- Manufacturer Name
- Operating System Name
- Operating System Version
- Product Name
- Product Version
- Product Language
- Application Type
- SHA-1 of file (digital fingerprint)
- MD5 of file
- CRC32 of file
- File Name
- File Size

The research environment allows NSRL to collaborate with researchers who wish to access the contents of the virtual library. Researchers may perform tasks on the NSRL isolated network that involve access to the copies of media, to individual files, or to "snapshots" of software installations. In addition to the media copies, NSRL has compiled a corpus of the 25,000,000 unique files found on the media, and examples of software installation and execution in virtual machines.

Trevor: Could you give us a brief overview of what exactly is the content of the library? What data and metadata do you collect and how do you work with it?

Doug: The library contains commercial software, both off-the-shelf shrink-wrapped physical packages and download-only "click-wrapped" digital objects. This includes computer operating systems, business software, games, mobile device apps, multimedia collections and malicious software tools.

Most of the software in the NSRL is purchased. We try to acquire everything the top selling lists. Some software we hear about by word of mouth, some by schedule (like tax programs each tax year, security, antivirus) and some by requests from law enforcement and other agencies. We accept donations from manufacturers and have paperwork to state we will not use the software license. We accept donations of used software as long as it is in useable condition but there is no guarantee that it will make it into the NSRL.

The data and metadata is detailed in documents on the NSRL website. To summarize, we collect accession data familiar to your readers; the information about the manufacturer and publisher, the minimal requirements listed, the number and types of media, etc. We also process the contents of the media to obtain metadata about the file system(s), directory structure, file types (based on signature strings) and many file-level metadata as I mentioned in the previous question.

NSRL makes minimal use of this metadata. We perform mock investigations using the metadata to measure the applicability. We investigate the randomness of the cryptographic algorithm results. We are constantly seeking related collections with which we could combine an index or translate a taxonomy, to cross-reference NSRL data with other sets.

Trevor: In the context of thinking about NSRL as a research environment it seems that the key value there is the corpus of software, the 23,809,431 unique files, that you have identified. Could you tell us about some of the research uses these have served so far? The audience for the blog varies widely in technical knowledge so it would be ideal if you could unpack these concepts a bit too.

Doug: The highest value, in my opinion, is the provenance and persistence of the collection. Given the virtual library, it is easy to apply new technology, new algorithms to the entire set or specific content automatically, while maintaining the relationship to previous work and the original media.

NSRL has applied several cryptographic algorithms against the corpus, and statistically analyzed the results. This is an interesting measurement of the algorithm properties within the relatively small scope of binary executable file types. NSRL found that indeed there were no collisions among the 25 million files.

Working with a collaborator, we are able to define precise, static content sections of executable files, obtain a digital fingerprint of those sections, then identify those sections when they are present on a running computer. This can allow an investigator to determine that a program was running, even though the files do not exist on the computer.

Working with a collaborator, we are able to provide practical feedback on the development of an algorithm called a similarity digest. Currently, if you have two digital copies of the Gettysburg Address text, one which begins “Five score and …”, the two cryptographic hashes of the differing files will be extremely dissimilar, as intended. Two similarity digest results on the two Address files will be similar, and the similarity can be measured. Algorithms of this kind are also known as “fuzzy” hashes, and they tend to be impractical for very large sets. We are assisting in developing a practical implementation.

NSRL has in past limited metadata collection to the content of the application media. We have now acquired the resources and defined the processes to automatically install an operating system on a virtual machine, run the OS, perform noteworthy tasks, install applications, generate content, uninstall applications, etc. This enables the collection of metadata on dynamic system files, registries, log files, memory, various versions of user-generated files. We

can use some of this metadata as feedback into our core process, and we have some research opportunities.

Another imminent collaboration is the creation of many word processing documents with created with different applications and multiple versions that contain the same text. A corpus of document tags or codes spanning versions and products has generated some interest.

Trevor: Could you tell us a little bit about the NSRL environment? What kinds of technologies and software are you currently using currently and what are you exploring for use in the future?

Doug: We are fortunate to have three contiguous rooms, one that houses the physical library, one that houses the data entry workstations, and one that houses servers and storage. The proximity of the rooms allowed us to pull our own cables, which makes that level of our infrastructure a controlled, known quantity.

The physical library has an alarmed, multi-factor entry control. The shelf system is a powered collapsing system which defaults to a closed, fire-retardant position. The environment is not kept within the recommended practices for archives; this was considered, but not implemented. Heat, fire, humidity and other risks are minimized to the best extent we can.

NSRL has strived to keep infrastructure implementations to hardware and technologies that can be quickly obtained and made functional in the event of a disaster. I would prefer to not name manufacturers at this time, but am willing to discuss those details with individuals.

In the second room, core work is performed using OpenSuSE Linux workstations for browser-based data entry and media copying. The Linux machines can be created in bulk or ad hoc using a net boot image. This room also contains a system used to perform software installations, so the NSRL can collect installed files, registry information and other artifacts of a running application. This room contains a computer attached to the internet on which NSRL downloads digital-only distributions of software. A photography stand and flatbed scanning stations are in this room, used to create digital photos of packaging, so these photos can be used for data entry and research instead of shelved material.

Movement of original packages and media is limited to the previous two rooms.

The third room is a computer server room with racks of equipment. The media copies are stored on a commercial, expandable network (currently 42TB) that is capable of access by Windows, Apple and Linux computers. We have several quad-core rack mounted servers that perform the automated distributed metadata collection tasks. A PostgreSQL database and an Apache webserver reside on one of the rack servers which is dedicated to these functions. The database is on local storage in that server.

The equipment described in the previous paragraph is duplicated, and that is the research environment. Media images, individual files, virtual machine slices and all databases are

backed up across a dedicated fiber connection to storage several buildings distant. Verification of critical files is performed nightly. We also periodically ship copies of the critical files to NIST Boulder, CO, campus.

The software we use is mostly written in Perl, with some PHP for the browser-based data entry. Reuse is key, as is flexibility; the NSRL code is essentially a wrapper or application interface which calls third-party tools to manipulate media, files or systems.

We have a quality assurance process that involves loading NSRL quarterly candidate releases into several third-party digital forensics tools, in each publishing cycle.

We don't anticipate substantial changes to our technology or software in the near future. If anything, we would revisit our internal database design, and address some issues that did not scale up as well as we expected.

Trevor: If other organizations have special collections would NSRL be interested in adding those collections to the reference library? If yes, what process would you suggest to someone interested discussing such an arrangement?

Doug: NSRL is very interested in pursuing loan arrangements with other institutions. Transfer of materials to NIST need not be a requirement. Please contact me, or any NSRL staff, via nsrl@nist.gov.

Trevor: Are there more research uses or ways that you think the NSRL could play a role in digital preservation work and research? Further, if any of the folks who follow this blog are interested in exploring doing research involving the software corpus what should they put together and how should they go about getting in touch with your team?

Doug: We are new participants in the community, so I believe we are still at the point of introducing ourselves. I am hopeful that uses may be identified as our capabilities and activities are made known. This blog is a step in that direction, and I thank you for this opportunity. Anyone with questions regarding research access should contact me.

Trevor: As a final question, could you tell us a bit about how the NSRL came about? One of the tricky parts of digital stewardship is establishing the value and need for building and maintaining collections and I think the story of the need and uses that the NSRL serves offers a powerful frame for thinking about the kinds of coalitions and common needs that digital stewardship initiatives work to support.

Doug: Prior to NIST involvement in digital forensics, Law enforcement identified the need for automated methods to review the large number of files in investigations involving computers. The FBI "Known File Filter" project supplied hash values of known files, the NDIC "Hashkeeper" project supplied hash values of installed files and of "known malicious" data files. Several commercial and open source tools existed that each used different hash values (CRC32, MD4, MD5, SHA-1)

Hash values were exchanged informally throughout the entire community via email, FTP sites, etc. Investigators had to know where to find hash sets; investigators had to judge the quality of the hash sets. There was no central, trusted repository, and there were open avenues for conflicts of interest.

NIST was contacted because of its history of impartiality in research and standards development. Among the benefits of this involvement were :

- NIST is an unbiased organization, not in law enforcement, not a vendor
- NIST can control quality of data
- NIST can provide traceability by retaining original software
- NIST can provide data in formats useful by many existing tools
- NIST has distribution mechanism in the Standard Reference Data service

The result of this is a data set that is court-admissible, a process that is transparent, and a collection open to researchers.

Challenges in the Curation of Time Based Media Art

INTERVIEW WITH MICHAEL MANSFIELD, ASSOCIATE CURATOR OF FILM AND MEDIA ARTS AT THE SMITHSONIAN AMERICAN ART MUSEUM BY JOSE (RICKY) PADILLA, LIBRARY OF CONGRESS, APRIL 9, 2013

This time in the Insights Interviews series we get the chance to speak with Michael Mansfield, an associate curator of film and media arts at the Smithsonian American Art Museum and representative in Smithsonian's Time Based and Media Art Conservation Initiative. Mansfield has contributed to exhibitions including *The Art of Video Games* , *Watch This: New Directions in the Art of the Moving Image*, and *Nam June Paik: Global Visionary*. I'm excited to get the chance to speak with Michael about his experience and insights on the curation of time based media art.

Ricky: Can you tell us a bit about your work at the Smithsonian American Art Museum? I would be particularly interested to hear about how your work connects with digital preservation.

Michael: I am the Associate Curator for Film and Media Art overseeing and organizing the permanent collection, acquisitions and exhibition practices for digital, electronic and moving image artworks. Part of this work includes developing best practices for the preservation of artworks and related archive material comprised of digital and electronic materials. Digital media is an increasingly important aspect of our cultural heritage, and at the moment, it plays two critical roles in the museum's initiatives. First, the institution is authoring its own digital tools to assist in preservation efforts around media artworks, both analogue and computer driven. And second, contemporary artists are authoring artworks using new and unique digital languages. These issues present significant challenges, but challenges that we are eager to respond to.

Ricky: Could you give us some examples of some of the pieces you have worked with? What is particularly challenging about working with time based media art? It would be ideal if you could talk us through challenges in working with particular pieces.

Michael: Time based artworks are complex. A particularly challenging characteristic of time based art is that any single artwork may exist in a multitude of forms. From a preservation perspective, the art object is both the physical components –which often mean an array of components – and the binary signature, which may include multiple assets. The artist's relationship to both 'materials' is really very important to understanding the artwork's place. As a curator of this collection, I need to ensure that the two remain compatible in perpetuity. One example might be Jenny Holzer's artwork ***For SAAM***.

It is a 28' tall, site specific, light column suspended in the museum's Lincoln Gallery. The column is comprised of 80,640 LEDs, managed by integrated circuits and attached to several

customized circuit boards. Jenny Holzer's texts – the content displayed on the column – exist as code running in DOS on an old computer laptop. There are eloquent relationships between her text based work, the code on the machine, and the visualization in the gallery. Managing the complexities of that on exhibit and in the collection is daunting.

Ricky: What lessons have you learned in working with this material? Further, do you think the lessons you've learned in this work transfer more broadly to preserving objects with software components?

Michael: There is a steep learning curve with artworks of this kind. It is difficult to build an accurate model for handling all time based art, because artworks vary so much from piece to piece. That's what makes them unique. But strategies we develop for handling one artwork certainly give us experience to draw from on the next. With ***For SAAM*** for instance, we're learning that the code and the components are of equal importance. It would be unwise to migrate either part to a more stable system without accounting for the other.

Ricky: I imagine that contributing to The Art of Video Games exhibition held last year presented some opportunities to understand some of the nuances of working with time based media art in software. Could you mention other exhibitions or works which allowed you to appreciate the unique challenges of this type of curation?

Michael: Yes, The Art of Video Games presented some fantastic learning opportunities around media art in software. That exhibition in one sense foregrounded the evolutionary changes in hardware, software and interactivity. And it explored how creativity arrived from within the rules of the material. On the opposite side of the coin, we are exhibiting artworks in our WatchThis! gallery that experiment with those rules, challenging the very behaviors of technology and sometimes intentionally breaking technology. This is a strategy some artists employ to uncover new ideas and/or better understand ourselves. But, caring for artworks that intentionally break the rules, or even visualize destruction, certainly presents its challenges from a preservation perspective. We have to preserve something so that it can be continuously destroyed.

Ricky: How does the construction of a "curatorial narrative" for an exhibition of time based media art differ from one for traditional art?

Michael: "Curatorial narratives" are a curious thing I suppose, and the material really shapes any exhibition. I think important elements to consider when developing an exhibition of time based media are simply time and space. Digital and time based media creates, or can create, a new performance space accessed by actors and unfolding in real time. Shaping the 'spatial' relationships between the artwork and the audience, actor or player can be very informative. In looking at art: revealing the behaviors of the artist, the behaviors of the media and the behaviors of the participant give us invaluable insight for understanding ourselves and the world we inhabit.

Ricky: I would be curious to know if there are any essays, papers or projects that you have looked to for insight on helping ensure future access to these works. If there are, I would love to hear what you found particularly useful about them.

Michael: Digital preservation among art collections is a very hot topic for museums and institutions at the moment. Organizations like museums are not known for being particularly nimble, but commercial changes in technology are really forcing the issue. So now there are incredibly smart people tackling these issues within them. A number of pan-institutional projects have been formed and they are generating great ideas published for public consumption, (notably something that institutions do very well). The Smithsonian has its own Time Based Media Art Conservation Initiative that is currently investigating models for trusted digital repositories used for documenting, storing and maintaining moving image artworks. There is also the *Variable Media Network* started by the Solomon R. Guggenheim Museum. And, of course there is *Matters in Media Art* resulting from a fantastic collaboration between the New Art Trust and its partner museums at Tate in the UK, Museum of Modern Art in New York and the San Francisco Museum of Modern Art. Projects like these are a really compelling and inspiring use of resources.

Ricky: What different groups at the Smithsonian Institution are working on preserving this kind of time based media art? I would be curious to hear a bit about the different players and the different roles that are emerging around this material.

Michael: One thing that is very clear about "time based art" is that it would be impossible for one individual to understand every aspect of the field in all its complexity. The Smithsonian is large and includes a number of independent collections. While each museum on campus handles their respective collections differently, we've come together around these issues and are finding new ways to leverage the tremendous institutional knowledge captured here. There is a joint, time based art conservation initiative that includes representatives not only from each museum, but each discipline within the museum. We have curators, registrars, conservators, technology specialists, mathematicians, engineers, archivists … collectively, we can tackle challenges facing our time-based art collections by communicating with truly knowledgeable experts in other fields.

Ricky: What areas do you feel we need more research or tools to support conserving this kind of material?

Michael: I'd like to find interesting ways to document the lifecycle of media artworks. This might be out of left field a bit, but artworks like this seem to live and breath in ways that are unique in the arts and unique in their time or historical place. They grow, or shrink. They respond to their surroundings. They physically evolve. They consume. They age. They die … In some cases they reproduce. Outside of the box, I think we might benefit from some creative, comparative research with animal sciences, through their documentation of life cycles. We can look at the tools used by zoos and their conservation practices with living specimens. How do they document natural behaviors of a living creature? Perhaps this might generate some new ideas for handling something like an artwork, something that is uniquely human.