http://doi.org/10.7207/twgn20-02

Preserving Software: Motivations, Challenges and Approaches

Sheila M. Morrissey

DPC Technology Watch Guidance Note

August 2020

DigitalPreservationCoalition

ISSN: 2048-7916

First published in Great Britain in 2020 by the Digital Preservation Coalition.

1 Why preserve software?

We care about preserving software both for its own sake, and because preserved software is a vitally important means of preservation – possibly the only means we might have to access other sorts of digital artefacts in the future.

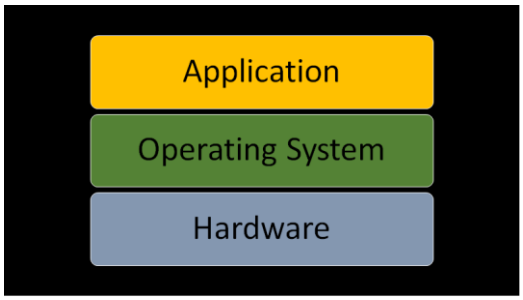
We preserve software to study techniques of creating it, the purposes to which it was put, and the conceptual, physical, social, and political systems within which it is created and executed (Ensmenger 2009). We inspect software, re-run it, re-write and re-configure it for many purposes: to make transparent its effect on us, to adapt it to new use, to ensure the reproducibility and integrity of research.

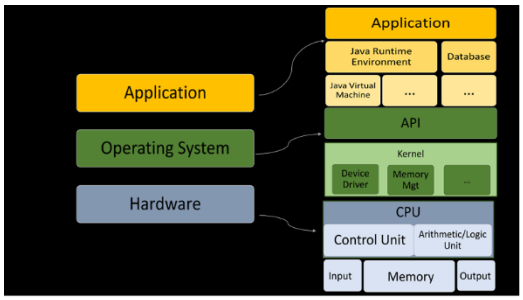
As a means of preservation of other objects, preserved software mediates the experience of all sorts of 'digital-native' artefacts: fine art, multi-media, hyper-linked 'books', single- and multi-player video games (Chue Hong 2019). Sometimes we access these artefacts via obsolescent software because we are scholars investigating the 'original experience' of these mediated artefacts. Increasingly, we are accessing them this way because it is the *only* experience we can have of these artefacts (Cochrane 2012; Morrissey 2010).

2 What are the technical challenges of preserving software?

Whether or not software is eating the world, it could well be said to be eating itself. Software, if not by intention the product of '*planned obsolescence*,' is certainly the very epitome of '*born obsolescent*.' Since its earliest days as a profession, software engineering has 'theorized' what is called the software life cycle. It has articulated laws of program evolution (including a Law of Continuing Change (Lehman 1980)) and the phases of that cycle, culminating in complete replacement at best; and, at worst, in abandonment. (Chapin at al., 2001; IEEE Standards Association 2006)

Further, both hardware and software are conceptualized, built, and executed in vertical 'layers' of increasing abstraction, as well as horizontal 'modules' of specialized functions and capabilities, with a myriad of interactions, up and down these layers, and back and forth across these components. These conceptual layers and modules are themselves subdivided into interacting layers and modules.





Then, at different times, components at each layer are swapped out for new ones – to fix a bug, perhaps, or to add new capabilities, or to make the component compatible with a change made in some other component that has been modified. Sometimes, other components at the same or higher or lower levels as these new ones 'break.' They cease to function, or function differently. When we try to reassemble these execution stacks years later, it is often difficult, if not impossible, to analyse and repair the cause of breakage (Dick and Volmar. 2018).

Details of configuration – parameters passed at runtime to an executable – are another artefact difficult to capture, anatomize, reproduce, or understand. This could be anything from settings in a configuration file on a local system, to personalization information stored on, and applied from, a remote server to a cell-phone or other application.

We face analogous complexities in saving source code. Source code is the text-file-based, human-readable code in which software is now almost entirely written. It is then translated via yet more software (a compiler) into the low-level ('machine') code, executed by a computer. We will want access to source code in the future for many of the same reasons we will want effective access to executable software. This means assuring preservation of correctly versioned compilers, along with appropriate software and hardware stacks. Even then, we face the problem of understanding, not just the syntax of the programming language used, but the larger coding idioms employed to organize code and communicate its organization and capabilities to human readers (Matthews et al., 2009; Morrissey 2010).

3 What are the legal challenges of preserving software?

There is virtually no software, including free/libre and open source software (FLOSS), that does not fall under one or another regime, national or supranational, of intellectual property (IP) law (Charlesworth 2012). Anyone preserving software and making that preserved artefact accessible for inspection or use must grapple with issues of license and copyright, sometime with conflicting imperatives from national or international law (Charlesworth 2012). It is, at best, unclear whether 'fair dealing' (in the UK) or 'fair use' (in the USA), would avert legal entanglements for institutions preserving software, even with such provision as is made for non-commercial research or private study (ARL 2018; Charlesworth 2012).

It is not clear what legal restrictions exist, under different legal IP regimes, on the text mining of source code repositories. Nor is it clear that software preserved for the purpose of rendering other preserved digital artefacts, even if legally obtained and preserved, could legally be used to provide such access.

As with books and other published material, software suffers from an 'orphan works' problem. Even with the best will in the world to conform with licenses and IP law, it might not be possible to establish the rights holder of a piece of software (Charlesworth 2012).

Further, preserving software often means grappling with the technological enforcement of digital rights management (DRM). The process of installing older software might include validation via a license key. Even if the conservator has the key, the validating service might no longer be in existence – and the software could be rendered unusable (ARL 2018; Charlesworth 2012). The Digital Millennium Copyright Act (DMCA) in the US, for example, prohibits any tampering with DRM technology.

4 What are the curatorial challenges of preserving software?

The first challenge of software curation is to determine, in light of one's own institutional mission, the 'what' of software curation. What is the focus of your curatorial activities? Candidates include the systematic collection of general-purpose systems, such as older operating systems; software and hardware of historical interest; digital-native artefacts of cultural or social significance (such as time-based media, digital arts, computer games); and source code, especially (but not exclusively) software used in scholarly research (Bearman 1987; Chassanoff et al., 2018).

Then there are the challenges of the 'how.' There is risk management, including taking proper care of any software containing sensitive or protected content, or that falls under the rubric of national security issues (Charlesworth 2012). As mentioned above, there are the risks associated with license and other IP constraints, including the necessity to determine any limitations on access to preserved software that might be required to conform to IP or other restrictions. There is the possibility of malicious code, including computer viruses, embedded in an acquisition, which could contaminate or damage a collection (Rosenthal 2015).

To be useful, curated software must be discoverable. How do we describe software effectively? The list of properties to be described will vary according to the purposes for which we collect software. These properties might include details of the software stack needed to be able to run it; bibliographic information and persistent identification for citation; key descriptive information about algorithms or languages employed. (Bettivia 2016; Katz et al., 2019; McDonough 2010; Smith et al., 2016) As yet, there is no single ontology or metadata vocabulary for software description, or indeed agreed-upon taxonomies for these and other possible sets of criteria.

Whether we are studying software as an artefact of interest or are collecting it as a means of access to other artefacts, contextual information and materials are crucially important both for understanding and for executing it. There are many possible candidates for the contextual material and information we should collect with the software, including material aspects (such as distribution media); user guides; reference manuals; postings to old listservs and electronic boards; and original hardware on which the software ran. Possible approaches to preserving context include documenting what a running program 'looks like,' using screen shots, videos, or automated session recorders, and noting what the operation of the program 'takes for granted' – including gestures such as swiping, or the use of auxiliary devices such as keyboards and mice (Bettivia 2016; Cranmer 2017; Depocas et al., 2003; Newman 2011).

As is perennially true for digital preservation, software preservation is still largely an unfunded mandate.

5 What are people doing now to preserve software?

Approaches to Technical Challenges

Just as Moliere's *bourgeois gentilhomme* had been speaking prose for over forty years while knowing nothing of it, so software engineers the world over have been practising software preservation for over half a century – having done so under the rubric of software maintenance. Notably, these 'maintainers' have succeeded in the prosaic task of keeping key government and financial systems running. The preservation community has this experience, and the scholarly literature created around it, to draw upon.

In business and government, this very old software typically does not run on original hardware. However, keeping software running as created, on original hardware, is the preservation approach taken by some institutions. These include the National Museum of Computing in the UK, and the Computer History Museum and the Living Computers Museum + Labs in the US.

Also from the commercial world, and enriched by the community of computer gaming enthusiasts, comes a crucial digital preservation strategy: emulation. Emulation is the use of software to mimic the behaviour (the 'instruction set') of one type of computer hardware while running an application on different computer hardware, with a different instruction set, to recreate, as closely as possible, the behaviour of the original hardware/software stack (Rosenthal 2015).

Emulation was proposed as a preservation strategy by Jeff Rothenberg in 1995 (Rothenberg 1995). A number of individuals and institutions have applied both general-purpose and purpose-built emulators to the challenge of digital preservation, including CAMiLEON[1], KB Dioscuri[2], the EU's KEEP Emulation Framework project (Anderson et al., 2010), and Internet Archive's software collection[3].

A key project in the application of emulation to preservation is the University of Freiburg's DPC award winning bwFLA[4]. bwFLA delivers emulation as a web service. It has been used, for example, by Rhizome to deliver some of the born-digital artworks in its collection. Also based on bwFLA is the current EaaSI (Emulation as a Service Infrastructure) project[5], which is building both a community and a shared infrastructure for capturing, storing, cataloguing, and serving up emulated software stacks (Cochrane 2019).

There are some challenges in employing emulation as a preservation solution. It is technically challenging to create an executable 'image' of the software stack to be run over the emulator. And the emulators, being themselves software, will themselves have to be maintained, so that they will continue to be usable as the hardware and software they run on continues to evolve.

Container technology is similar to emulators, though more lightweight. A preserved software image and its software dependencies are stored together as a 'container' and run on the same host operating system and hardware instruction set used by software when it was created. Container technology is the basis of such tools as ReproZip and commercial service Code Ocean, which aim to capture the stack of software dependencies used in running an instance of software developed in connection with scholarly research. Containers are however less robust over the very long term from a preservation point of view, as they depend on the continued availability of the original operating system and hardware instruction set. (Boettiger 2015; Emsley and DeRuore 2017)

The not-for-profit Software Heritage (SH) has undertaken to collect, preserve, and share the source code of all publicly available software (Di Cosmo and Zacchiroli 2017). Using both automated and manual methods, SH harvests source code from the public repositories currently or previously used by developers, such as GitHub, BitBucket, Debian, Google Code, and GNU.

Cern, as a participant in the EU-funded, open-access/open-data OpenAIRE project, established the Zenodo repository. Among the digital research artefacts it is intended to preserve is software source code. To that end, Zenodo has developed a software tool that automates the creation of a Digital Object Identifier (DOI) for software that developers store on the GitHub's commercial source code repository, and the replication of that software to Zenodo for long-term preservation.

Approaches to Legal Challenges

While some technical approaches to preserving software are promising results, the legal challenges are proving more stubbornly resistant to solution.

The principle of fair use has made for some leeway in the US, where a successful petition by the Software Preservation Network (SPN), Harvard University's CyberLaw Clinic, the American Library Association (ALA), the Association of Research Libraries (ARL), and the Association of College and Research Libraries (ACRL) to the US Copyright Office resulted in a three-year exemption from DMCA

[1] https://web.archive.org/web/20060909234230/http://www.si.umich.edu/CAMILEON/index.html
[2] https://web.archive.org/web/20091227142651/http://dioscuri.sourceforge.net/
[3] https://archive.org/details/software
[4] http://eaas.uni-freiburg.de/
[5] https://www.softwarepreservationnetwork.org/projects/emulation-as-a-service-infrastructure/

provisions against the circumvention of DRM technology for libraries, archives, and museums. SPN has published guidelines for preservationists on the applicability of that exemption (Albert and Lee 2018), complementing the 'Code of Best Practices in Fair Use for Software Preservation.' (ARL 2018).

Fair use and the DMCA exemption are of course only applicable in the US. In the 2018 'Paris Call: Software Source Code as Heritage for Sustainable Development' issued by UNESCO, Inria, and Software Heritage, the signatories noted the need to 'ensure necessary exceptions to copyright and limitations on intermediary liability related to software for archival preservation, accessibility, education and research purposes.'[6] Active efforts by SH and others resulted in an exclusion of software source code from the copyright provisions of the EU Directive on Copyright in the Digital Single Market.

Approaches to Curation Challenges

Collaborative effort is proving key both to raising awareness of the importance of software preservation, as well as accomplishing some specific curatorial aims.

Software Heritage, along with Force-11 and the Research Data Alliance (RDA) Source Code Interest Group, have worked jointly on developing identification schemas for source code. They, along with Software Preservation Network, have provided both guidance and tools for software citation, increasing the profile of research software. This work is reinforced by those in the research community concerned with reproducibility of research results – efforts which include for example formal software artefact review and badging for an increasing number of Special Interest Groups (SIGS) and conferences of the Association for Computing Machinery (ACM) (Marquis et al., 2016).

Part of Software Heritage's collection strategy is the harvesting of source code metadata available in the configuration files of the various source code management (SCM) systems used by commercial and other repositories. They have developed software to create metadata compliant with CodeMeta. The CodeMeta project includes crosswalk tools from many of the most common SCM vocabularies.[7]

Along with automated tools for gathering software metadata, museums and galleries have begun collaborating with artists (both at the time of creation and at donation), to gather relevant contextual information, including execution stack, intended rights constraints, and also information about the intended affect of the art work (Cranmer et al., 2017; Depocas et al., 2003; Newman 2011; Verbruggen 2018). The UK's National Videogame Archive is preserving 'walkthrough texts.' These are player-produced artefacts (typically plain text) that record techniques of proceeding through the various possible paths of video games, as well as ways to experiment, for example by means of bugs in the game software, with ways of forging new game paths and experiences (Newman 2011). Libraries, museums, and archives have also begun talking directly with commercial software vendors, not simply to obtain donations of software, but also, where necessary, to obtains releases on existing license and copyright constraints.

6 References

Albert, Kendra and Lee, Kee Young. (2018) *A Preservationists Guide to the DMCA Exemption for Software Preservation*. Software Preservation Network. Available at

[6] https://unesdoc.unesco.org/ark:/48223/pf0000366715.locale=en
[7] https://codemeta.github.io/

https://www.softwarepreservationnetwork.org/a-preservationists-guide-to-the-dmca-exemption-for-software-preservation/.

Anderson, David, Delve, Janet, and Dan Pinchbeck. (2010) 'Toward A Workable Emulation-Based Preservation Strategy: Rationale and Technical Metadata.' *New Review of Information Networking*, 15(2) 2, pp. 110-131. Available at https://www.tandfonline.com/doi/abs/10.1080/13614576.2010.530132?src=recsys&journalCode=rinn20. DOI: 10.1080/13614576.2010.530132.

The Association of Research Libraries (ARL) (2018, rev 2019). *Code of Best Practices in Fair Use for Software Preservation.* Available at https://www.arl.org/resources/code-of-best-practices-in-fair-use-for-software-preservation/

Bearman, David. Collection Software: A New Challenge for Archives & Museums, Pittsburgh: Archives & Museum Informatics, 1987. Available at http://www.archimuse.com/publishing/bearman_col_soft.html

Bettivia, Rhiannon. (2016) 'Where Does Significance Lie: Locating the Significant Properties of Video Games in Preserving Virtual Worlds II Data.' *International Journal of Digital Curation*, 11(1), pp. 17–32. Available at http://dx.doi.org/10.2218/ijdc.v11i1.339. DOI: 10.2218/ijdc.v11i1.339.

Boettiger, Carl. 'An introduction to Docker for reproducible research.' (2015) *SIGOPS Oper. Syst. Rev.* 49(1), pp. 71-79. Preprint available at https://arxiv.org/abs/1410.0846. DOI: 10.1145/2723872.2723882

Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F. and Tan, W. (2001) 'Types of software evolution and software maintenance.' *J. Softw. Maint. Evol.: Res. Pract.*, 13(1), pp. 3-30. Available at https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.220. doi:10.1002/smr.220.

Charlesworth, Andrew. (2012) Intellectual Property Rights for Digital Preservation: DPC Technology Watch Report 12-02. Available at http://dx.doi.org/10.7207/twr12-02. DOI:10.7207/twr12-02.

Chassanoff, Alexandra, Al Noamany, Yasmin, Thornton, Katherine, and John Borghi. (2018) 'Software Curation in Research Libraries: Practise and Promise.' *Journal of Librarianship and Scholarly Communication,* 6 (General Issue), eP2239. Available at http://doi.org/10.7710/2162-3309.2239. DOI: 10.7710/2162-3309.2239.

Cochrane, Euan. (2012) Rendering Matters, Archives New Zealand. Available at https://web.archive.org/web/20190122170515/http://archives.govt.nz/rendering-matters-report-results-research-digital-object-rendering.

Cochrane, Euan, Rechert, Klaus, Anderson, Seth, Meyerson, Jessica, and Ethan Gates. (2019) 'Towards a Universal Virtual Interactor (UVI) for Digital Objects.' *Proceedings of the 16th International Conference on Digital Preservation iPRES 2019*. Available at https://ipres2019.org/static/pdf/iPres2019_paper_128.pdf. DOI:10.17605/OSF.IO/AZEWJ.

Cranmer, C. (2017) Preserving the emerging: virtual reality and 360-degree video, an internship research report. Netherlands Institute for Sound and Vision. Available at http://publications.beeldengeluid.nl/pub/584

Depocas A., Ippolito J., Jones C. (editors). (2003) "Permanence Through Change: The Variable Media Approach.' The Solomon R. Guggenheim Foundation, New York, and The Daniel Langlois Foundation

for Art, Science, and Technology, Montreal. Available at https://variablemedia.net/e/preserving/html/var_pub_index.html.

Di Cosmo, Roberto, and Zacchiroli, Stefano. (2017) 'Software Heritage: Why and How to Preserve Software Source Code.' *Proceedings of iPRES 2017 - 14th International Conference on Digital Preservation, Sep 2017, Kyoto, Japan*., pp.1-10. Available https://ipres2017.jp/wp-content/uploads/19Roberto-Di-Cosmo.pdf.

Dick, S. and D. Volmar. (2018) 'DLL Hell: Software Dependencies, Failure, and the Maintenance of Microsoft Windows.' *IEEE Annals of the History of Computing*, 40(4), pp. 28-51. Available at https://ieeexplore.ieee.org/document/8509170. DOI: 10.1109/MAHC.2018.2877913

Emsley, Iain and DeRuore, David. (2017) 'A Framework for the Preservation of a Docker Container.' *International Journal of Digital Curation*, 12(2), pp. 125-135. Available at http://dx.doi.org/10.2218/ijdc.v12i2.509. DOI: 10.2218/ijdc.v12i2.509.

Ensmenger, N. (2009) 'Software as History Embodied' *IEEE Annals of the History of Computing*, 31(1), pp. 88-91. Available at https://doi.ieeecomputersociety.org/10.1109/MAHC.2009.16. DOI 10.1109/MAHC.2009.16.

Chue Hong, Neil. (2019) 'From shoeboxes to software preservation.' Presentation at Insert Coin to Continue: DPC Briefing Day on Software Preservation. 7th May 2019. https://doi.org/10.6084/m9.figshare.8088290.

IEEE Standards Association. (2006) *14764-2006 - ISO/IEC/IEEE International Standard for Software Engineering - Software Life Cycle Processes - Maintenance*. Available at https://standards.ieee.org/standard/14764-2006.html.

Katz DS, Bouquin D, Hong NPC, Hausman J, Jones C, Chivvis D, et al. (2019) "Software Citation Implementation Challenges." Available at https://arxiv.org/abs/1905.08674.

Lehman, M. M. (1980) "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle." *Journal of Systems and Software*, 1, pp.213-221. Available at https://doi.org/10.1016/0164-1212(79)90022-0.

Marquis, M., Hall, L., Hemami, S., Setti, G., Forster, M., Grenier, G., … Moore, K. (2016). *Report on the First IEEE Workshop on The Future of Research Curation and Research Reproducibility*. Available at http://www.ieee.org/publications_standards/publications/ieee_workshops/ieee_reproducibility_workshop_report_final.pdf

Matthews, B., Shaon, A., Bicarregui, J., Jones, C., Woodcock, J., and Conway, E. (2009) 'Towards a Methodology for Software Preservation.' *Proceedings of 6th International Conference on Preservation of Digital Objects (iPres 2009)*. Available at https://escholarship.org/uc/item/8089m1v1.

McDonough, J., Olendorf, R., Kirschenbaum, M., Kraus, K., Reside, D., Donahue, R., Phelps, A., Egert, C., Lowood, H., & Rojo, S. (2010). *Preserving Virtual Worlds Final Report*. http://hdl.handle.net/2142/17097

Morrissey, Sheila. (2010) 'The Economy of Free and Open Source Software in the Preservation of Digital Artifacts.' Library Hi Tech, 28(2), pp.211 – 223. Available at https://doi.org/10.1108/07378831011047622. DOI:10.1108/07378831011047622.

Newman, James. (2011) '(NOT) Playing Games: Player-Produced Walkthroughs as Archival Documents of Digital Gameplay', International Journal of Digital Curation 6(2), pp. 109-127. Available at: https://doi.org/10.2218/ijdc.v6i2.206. DOI: 10.2218/ijdc.v6i2.206.

Rieger, O.Y., Murray, T., Casad, M., Alexander, D., Dietrich, D., Kovari, J., Muller, L., Paolillo, M. & Mericle, D.K. (2015) 'Preserving and Emulating Digital Art Objects.' Available at: https://ecommons.cornell.edu/bitstream/handle/1813/41368/PAFDAOWhitePaperCornell.pdf?sequence=5

Rosenthal, David S.H. (2015). 'Emulation & Virtualization as Preservation Strategies.' The Andrew W. Mellon Foundation. Available at https://mellon.org/Rosenthal-Emulation-2015.

Rothenberg, Jeff. (1995) 'Ensuring the Longevity of Digital Documents.' Scientific American, 272(1). Revision 1999, available at https://www.clir.org/wp-content/uploads/sites/6/ensuring.pdf

Smith, AM, Katz, DS, Niemeyer, KE & Chue Hong, N 2016, 'Software citation principles', PeerJ Computer Science, vol. 2, no. 86. Available at https://doi.org/10.7717/peerj-cs.86. DOI: 10.7717/peerj-cs.86.

Verbruggen, Erwin. (2018) 'Preserving Interactives.' White Paper, Netherlands Institute for Sound and Vision. Available at https://www.prestocentre.org/system/files/library/resource/14-6_preservering_interactives_whitepaper_e_verbruggen_1.pdf.