HAL open science

Preserving landmark legacy software with the Software Heritage Acquisition Process

Laura Bussi, Roberto Di Cosmo, Carlo Montangero, Guido Scatena

To cite this version:

Laura Bussi, Roberto Di Cosmo, Carlo Montangero, Guido Scatena. Preserving landmark legacy software with the Software Heritage Acquisition Process. iPres2021 - 17th International Conference on Digital Preservation, Oct 2021, Beijing, China. hal-03375572

HAL Id: hal-03375572

https://hal.science/hal-03375572v1

Submitted on 12 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Preserving landmark legacy software with the Software Heritage Acquisition Process

Laura Bussi
University of Pisa
Italy
laurabussi@live.it

Roberto Di Cosmo
Inria and University of Paris
France
roberto@dicosmo.org
0000-0002-7493-5349

Carlo Montangero
University of Pisa
Italy
carlo@montangero.eu

Guido Scatena
University of Pisa
Italy
guido.scatena@unipi.it

Abstract – The source code of landmark software developed since the beginning of the computer era is a precious part of our cultural heritage, and needs to be properly rescued, curated, archived and made available to present and future generations. In this article, we present the Software Heritage Acquisition Process, that has been designed to provide detailed guidelines on how to perform this important task, preserving important historical information. This process has been validated extensively on several important pieces of software source code of historical relevance in the University of Pisa, in collaboration with UNESCO, and is open to all for adoption and improvement.

Keywords – software preservation, legacy software, source code, acquisition process

Conference Topics – new developments; capacity building

I. Introduction

Software binds our personal and social lives, embodies a vast part of the technological knowledge that powers our industry, supports modern research, mediates access to digital content and fuels innovation. In a word, growing part of our collective knowledge is embodied in, or depends on software artifacts.

Software is written by humans, in the form of software Source Code, a precious, unique form of knowledge that, besides being readily translated into machine-executable form, should also "be written for humans to read" (Abelson and Julie Sussman [1]), and "provides a view into the mind of the designer" (Shustek [2]).

As stated in the Paris Call on Software Source code as Heritage for sustainable development (Report [3]), from the UNESCO-Inria expert group meeting, it is essential to preserve this precious technical, scientific and cultural heritage over the long term.

Software Heritage is a non-profit, multi-stakeholder initiative, launched by Inria in partnership with UNESCO, that has taken over this challenge. Its stated mission is to collect, preserve, and make readily accessible all the software source code ever written, in the Software Heritage Archive. To this end, Software Heritage designed specific strategies to collect software according to its nature (Abramatic, Di Cosmo, and Zacchiroli [4]).

For software that is easily accessible online, and that can be copied without specific legal authorizations, the approach is based on automation. As of February, Software Heritage has already archived more than 6 billion unique source code files from over 90 million different origins, focusing in priority on popular software development platforms like GitHub and GitLab and rescuing software source code from legacy platforms, such as Google Code and Gitorious that once hosted more than 1.5 million projects.

For source code that is not easily accessible online, a different approach is needed. It is necessary to cope with the variety of physical media where the source code may be stored, the multiple copies and versions that may be available, the potential input of the authors that are still alive, and the existence of ancillary materials like documentation, articles, books, technical reports, email exchanges. Such an approach shall be based on a focused search, involving a significant amount of human intervention, as demonstrated by the pioneering works reconstructing the history of Unix (Spinellis [5]) and the source code of the Apollo Guidance Computer (Burkey [6]).

In order to rescue, curate and illustrate landmark legacy software source code, a joint initiative of Software Heritage and the University of Pisa, in collaboration with UNESCO, has led to developing SWHAP, the ***SoftWare Heritage Acquisition Process***, that provides detailed, ac-

17th International Conference on Digital Preservation
iPRES 2021, Beijing, China.
Copyright held by the author(s). The text of this paper is published under a CC BY-SA license (https://creativecommons.org/licenses/by/4.0/).
DOI: 10.1145/nnnnnnn.nnnnnnn

tionable guidelines now available for everyone.

This article presents the SWHAP process, focuses on relevant aspects of its implementation during the legacy software rescue campaing carried on in Pisa during 2019, and provides examples of landmark legacy software that have been handled using SWHAP on this occasion.

II. The acquisition process, an abstract view

This section, extracted from the full SWHAP guidelines [7], describes the acquisition process for software artifacts at an *abstract* level, that is, without making specific assumptions on the *tools, platforms and technologies* that may be used to perform the various operations described here.

A. Phases

The activities involved in the acquisition process can be organized in the following four phases, of which the first one is *conservative*, i.e., it is devoted to save the raw materials that the other phases will build upon.

Figure 1 provides a pictorial view of the process, its phases, data stores and roles.

1. Collect

The purpose of this phase is *to find* the source code and related materials and *gather* it *as is* in a physical and/or logical place where it can be properly *archived* for later processing.

Various *strategies* are possible for collecting the raw materials: a dedicated team may proactively search for the artifact of specific software that has been identified as relevant (*pull approach*), or a crowdsourcing process may be set up to allow interested parties to submit software that has not been previously identified (*push approach*).

Source code can be provided in a *digital* or *physical* form. Typically, source code for old machines (such as the first Italian computer, CEP [8], now exposed in the Pisa museum of computing) is available only as paper printouts that may even include hand-written comments: all these materials deserve to be preserved.

Related materials can include research articles, pictures, drawings, user manuals: all of these are part of the software history and need to be preserved as well as the source code.

At this stage of elaboration of the process, this phase is better thought of as *abstract*, in the sense that several, more focussed descriptions should be provided to cater for the different situations identified. The same applies to the Curator role, which may need different capabilities in different scenarios.

2. Curate

The purpose of this phase is *to analyze, cleanup and structure* the raw materials that have been collected.

Preparing software source code for archival in **Software Heritage** requires special care: the source code needs to be *cleaned up*, different *versions* with their *production dates need to be ascertained*, and the *contributors need to be identified* in order to build a *faithful history of the evolution* of the software over time.

Also, proper *metadata* should be created and made available alongside the source code, providing all the key information about the software that is discovered during the curation phase. We recommend to use the vocabulary provided by **CodeMeta** as an extension to schema.org (see https://codemeta.github.io/terms/); this includes the software runtime platform, programming languages, authors, license, etc.

Particular care is required to *identify the owners* of the different artifacts, and *obtain if needed the necessary authorizations* to make these artifacts publicly available[1].

3. Archive

The purpose of this phase is to contribute the curated materials to the infrastructures specialized for each kind of materials: ***Software Heritage*** for the *source code*, ***Wikimedia*** for *images or videos*, ***open access repositories*** for *research articles*, ***Wikidata*** for *software descriptions and properties*, and so on.

Well established guidelines are available for contributing materials to Wikimedia (see [10]) and Wikidata (see [11]), hence we focus primarily on curating and contributing the software source code to Software Heritage, a process that is new and may require rather technical steps.

4. Present

The purpose of this phase is to create dedicated presentations of the curated materials.

Once the curated materials are made available in the dedicated infrastructures, it is possible to use it to create presentations for a variety of purposes: special events, virtual or physical expositions for museums or websites.

For this, the archived materials need to be referenced using the identifiers that each platform provides for its contents. Software Heritage provides intrinsic persistent identifiers that are fully documented at https://docs.softwareheritage.org/devel/swh-model/persistent-identifiers.html

The presentation phase is out of the scope of this document, and as such we are currently not providing a supporting implementation. Anyway, a good example

[1] This is a complex issue, that may need to be handled according to country-specific regulations and is out of the scope of the present document. In the United States, one may leverage the "fair use" doctrine, see for example the detailed analysis presented in [9].

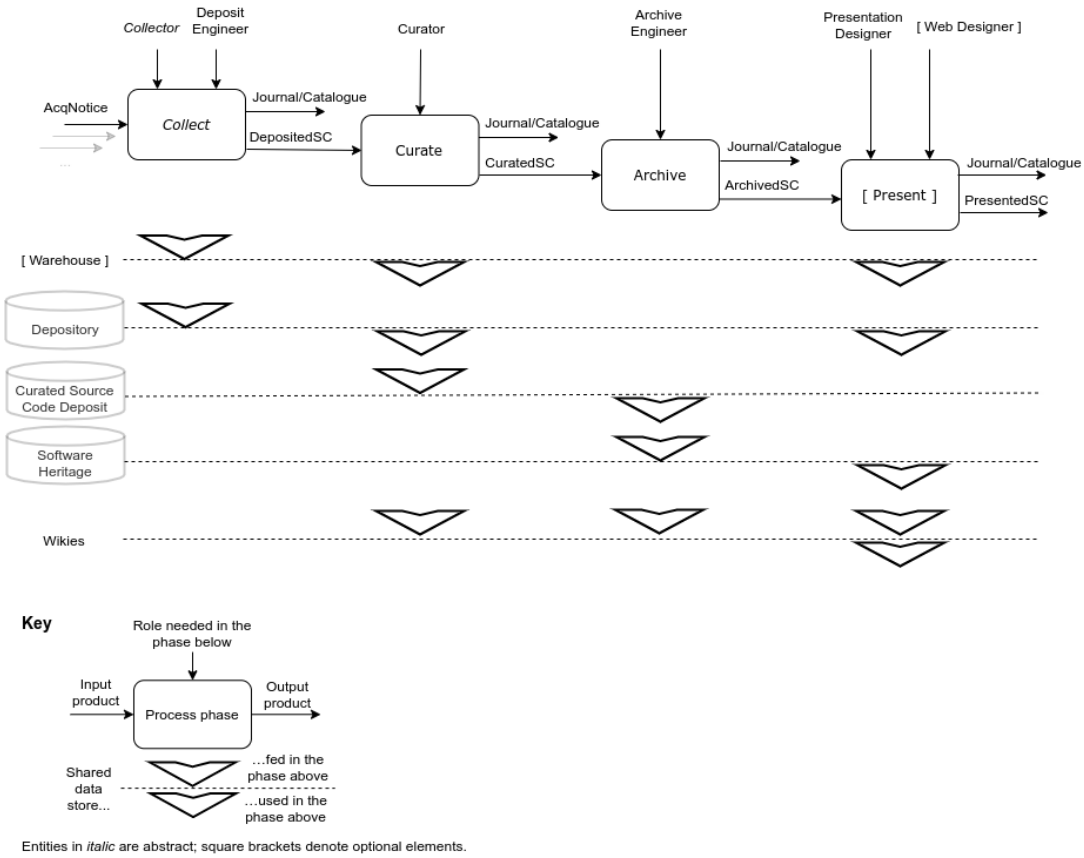


Figure 1: Source code acquisition process.

of what can be done is the https://sciencestories.io website.

B. *An iterative process*

New information may arise at any time: new raw materials may be discovered, refined information may be identified that needs to be added to the curation, and mistakes may need to be corrected. Hence, the overall process must be seen as *iterative*, in the sense that, when new data are available, the pertinent phase can be re-entered and the process enacted once more from there to update all the relevant information. This suggests that, whenever possible, the data stores should be fully versionable, not to loose historical information about the acquisition process itself.

C. *Resources needed by the process*

As any process supported digitally, SWHAP needs both human and technical resources to be enacted.

First of all, several data stores and working areas are needed, to save and make public the intermediate products, which are themselves of value, as already mentioned, and to pass the collected information across the phases. These are shown in the lower part of Figure 1, and are summarized here.

1. *Warehouse*

A physical location where physical raw materials are safely archived and stored, with the usual acquisition process[2].

2. *Depository*

A virtual space where digital raw materials are safely archived. The raw digital materials found in the Depository are used in the Curation phase to produce the source code that Software Heritage can ingest in the Archive phase.

The Depository holds also the related raw materials that may be elaborated and deposited in locations like WikiData, WikiMedia etc. - referred to as **Wikies** in fig. 1 - in the other phases.

3. *Workbench*

Any implementation of the process will need a virtual space and working environment where the activities can be carried out, with support for temporary storage and for logging the various operations in a journal.

[2] See for example https://collectionstrust.org.uk/spectrum/.

4. Curated source code deposit

A fully versioned repository, holding the reconstructed development history of the source code, in view of its transfer to Software Heritage.

5. Catalogues and journals

As shown in fig. 1, according to the best practices of the archival sciences, each phase shall produce both a *Catalogue* of its products and a *Journal* recording its activities - *who did what, and when*. A list of the *Actors* involved in the process is also necessary. Provision to store all these information safely has to be foreseen in any supporting implementation.

D. Roles in the process

With respect to the human resources, several roles are needed to enact the process, as indicated in the top part of fig. 1. Here is a short summary of the involved capabilities.

1. Collector

Searches and receives the raw materials. Identifies, classifies and separates source code and ancillary materials.

2. Deposit engineer

Masters the procedures to archive physical and digital materials, in the local context.

3. Curator

Prepares the version history, identifying the authors and other contributors. Provides a context to the source code, choosing among the ancillary materials.

4. Archive engineer

Masters the procedures to transfer the curated source code to SWH and to publish the context in the Wikies.

5. Presentation designer and Web engineer

These are out of the scope of this document, and are mentioned only to note that, though most of the presentations of the archived software will be on line, the abilities to design the contents of a presentation should be considered separately from the technical ones.

E. Implementation requirements

The abstract process may be implemented using different tools, platforms and technologies, as long as the following key requirements are satisfied.

1. Long term availability

The places where the artefact (both raw and curated) are stored must provide sufficient guarantees of availability over the long term. These places may be physical (warehouses), or logical (depositories).

2. Historical accuracy

Any supporting implementation should support the faithful recording of the authorship of the source code as well as of the reconstruction process, e.g., via a flexible versioning system.

3. Traceability

It must be possible to trace the origin of each of the artifacts that are collected, curated and deposited. For physical materials, we refer to common practice[3]. For digital artifacts, it is recommended to keep a *journal of all the operations* that are performed, and to automate them as much as possible, as the collection and curation process may require several iterations.

4. Openness

Any supporting implementation should be based on open and free tools and standards.

5. Interoperability

Any supporting implementation should provide support for the cooperation and coordination of the many actors playing the many roles of the acquisition process.

III. The SWHAP process for University of Pisa

At the University of Pisa, the SWHAP process has been implemented leveraging modern tools and platforms to manipulate software source code.

We provide here some key highlights of this particular implementation of the process, and we refer the interested reader to the complete guidelines [7] for full details, and a complete walkthrough on a medium sized example.

A. Instantiating the working areas

After an extensive study of the available options, the popular version control system Git [12] has been choosen as the designated tool for traceability and historical accuracy: it naturally provide a means to record the history of the modifications made to the digital assets, with information on *who did what and when*, and it is particularly well suited for managing complex workflows.

Once this choice has been made, it was natural to look for a way to use the same tool in all the storage and working areas, so the collaborative platform GitHub has been selected to instantiate the Depository, the Curated source code deposit and the Workbench.

The (logical) areas described above are then instantiated by means of *repositories* in GitHub. There are three repositories for each source code acquisition, one for each area of the abstract process:

[3] See for example in https://collectionstrust.org.uk/spectrum/.

Workbench repository, to implement the Workbench, i.e. a working area where one can temporarily collect the materials and then proceed to curate the code;

Depository repository, to implement the Depository, where we can collect and keep separated the raw materials from the curated source code;

Source Code repository, to implement the Curated source code deposit, where we store the version history of the code; this version history is usually "synthetic", rebuilt by the curation team, for old projects that did not use a version control system.

B. *Curating the source code version history*

While the full details of the concrete process used at Pisa can be found in the complete guidelines [7], we would like to focus now on some important aspects of the *curation* phase that leads to the final source code repository with the synthetic history.

When rescuing legacy software, one often finds several versions that have been released in the past, and the curation team needs to ascertain *for each version of the software* at least the *main contributing author* and the *exact date* of the release of this particular version.

The SWHAP guidelines recommend to collect all this information in a dedicated metadata file, named `version_history.csv`, that contains the following fields:

directory name name of the directory containing the source code of this version

author name name of the main author

author email email of the main author, when available

date original original date when this version was made

curator name name of the curator person or team

curator email the reference email of the acquisition process

release tag a tag name if the directory contains a release, empty otherwise

commit message text containing a brief note from the curation team

For traceability, the name of the main author and the date of the original release are of paramount importance, and must be properly preserved also in the final synthetic repository, and distinguished from the information related to the curation team.

The good news is that modern version control systems like Git allow to attach to every modification of the version history *two different sets* of metadata: one that describes the *author* of the modified digital assets, together with the *author date* of the modification, and another set that describes the *committer*, i.e. the person that adds the modified digital assets to the version history, together with the *committer date*.

Leveraging this functionality, it is possible to a synthetic version history of the development of a software project over time that clearly distinguishes the role of the original author, whose name and dates will go in the *author* field, from the role of the curator, whose name and date will go in the *committer* field.

An example of the result of the process is shown in Figure 2, that shows in the Software Heritage archive the release 1.1 of the CMM garbage collector that was rescued according to the SWHAP process in Pisa. We can clearly see that this version of the software has as main author *Giuseppe Attardi*, that, as ascertained during the curation process, *released it on October 27 1994*, while the synthetic repository that contains this particular version among many others *has been created by the Curation Team on December 11 2019*.

The fact that this information is stored safely in the version control system, and is properly presented in the Software Heritage archive, further supports the choice of using Git as the common tool for keeping track of the modifications.

We insist here on the need to carefully follow the process described in the full SWHAP guidelines to properly handle these metadata. By default, Git replicates the committer information into the author information, and a naive use of the tool may lead to incorrect results: on GitHub it's quite common to find today legacy source code with wrong metadata containing only the name of the person that uploaded the code on GitHub, and the date of the upload, while the real author name and the original date are missing[4].

IV. Results from enacting SWHAP in Pisa

In this section, we report on some of the legacy landmark projects whose software source code has been rescued, curated and archived in Software Heritage following the SWHAP process, as part of the activities to celebrate the 50th anniversary of the first course in Computer Science in Italy, established in Pisa in 1969.

That the Italian formal education in computing started in Pisa, was not by chance, and it seems useful to report briefly here on the tradition in the discipline in Pisa, to provide the context for what follows.

Starting March 1955, Pisa saw the construction of the first electronic computing machines conceived and designed in Italy, by Italian scientists and technicians. In-

[4] See for example the collection of legacy videogame source code found on https://github.com/videogamepreservation/

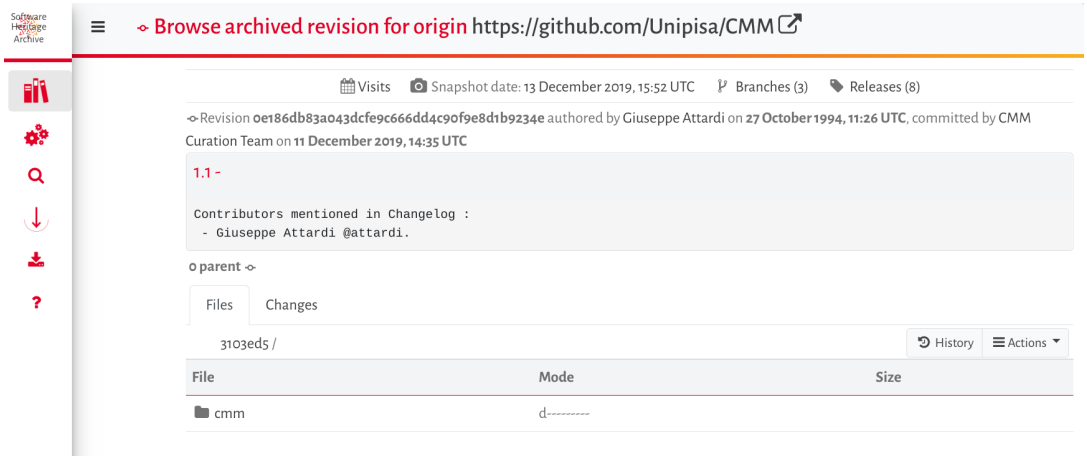


Figure 2: Distinguishing *author metadata* from *curator metadata*

deed, the University of Pisa then established the *Centro Studi Calcolatrici Elettroniche* (CSCE, Italian for Center for the Study of Electronic Computing machines) to exploit funds offered by the public administrations of Livorno, Lucca and Pisa. More funds came from Olivetti, then the major Italian manufacturer of electromechanical calculators, when the company CEO, Adriano Olivetti, signed an agreement with the University, as part of his strategy to enter the market of electronic computers.

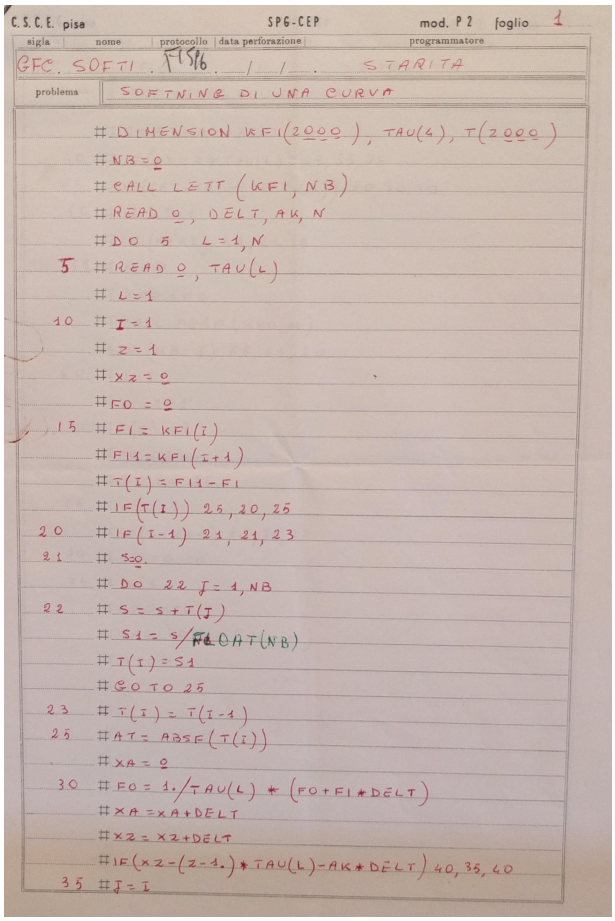
Contemporarily, the other notable efforts in Italy to keep up with the new field of electronic computing resorted to buying available machinery: in Milan, the local Polytechnic acquired a CRC 102A from the US firm National Cash Register; in Rome, the *Istituto Nazionale per le Applicazioni del Calcolo* (INAC, National Institute for Computing Applications) acquired what became known as FINAC, a Mark I* of the English firm Ferranti.

The joint venture in Pisa gave rise to two cooperating lines of design and development. Olivetti set up the *Laboratorio Ricerche Elettroniche* (LRE, Laboratory for Electronics Researches) in Barbaricina, at the outskirts of Pisa, to develop a general purpose commercial data processing machine; people at the CSCE were interested in a high precision high speed machine for scientific computing. In a couple of years, both teams delivered. The LRE presented the so called *Macchina Zero* (Machine #0) to Adriano Olivetti in July 1957. Eventually, this prototype evolved in the ELEA 9003, distributed by Olivetti since 1959 in more than 180 exemplars. Also in July 1957, the CSCE delivered an experimental machine, dubbed *Macchina Ridotta* (MR, Reduced Machine), which, despite its experimental nature, offered its computational services to the scientific community in Pisa until March 1959, when it was dismantled and partly recycled into a new machine, the *Calcolatrice Elettronica Pisana* (CEP, Pisa Electronic Computing machine). The CEP became operational in February 1961 and was in use till the Spring of 1970, when it was dismantled and eventually exhibited in the *Museo degli Strumenti per il Calcolo* (MSC, Computing Machinery Museum) of the University of Pisa.

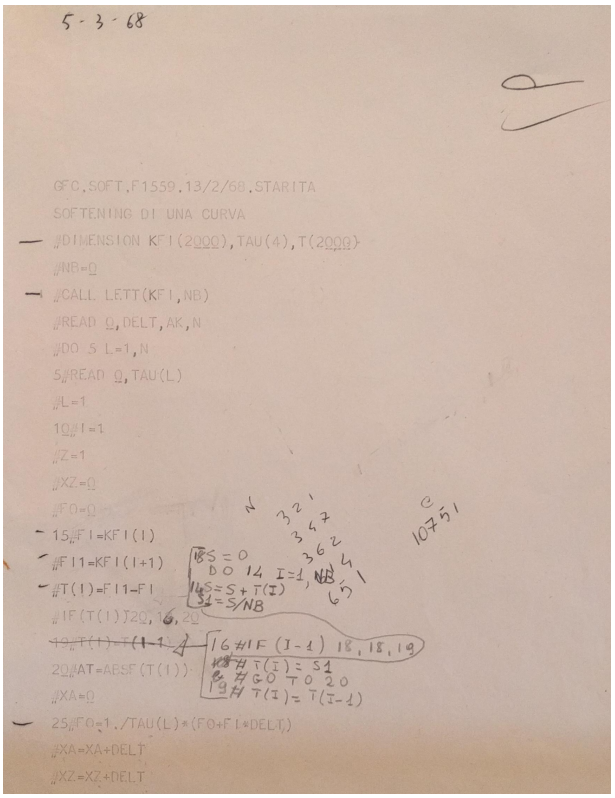
As foreseen by those who supported the initial choice of building a machine rather than buying one, the CEP project created a large amount of qualified expertise that fostered many new activities in Pisa. The CSCE became part of the *Consiglio Nazionale delle Ricerche* (CNR, National Research Council) under the name of Istituto di Elaborazione dell'Informazione (IEI, Information Processing Institute), and the University was able to establish the *Centro Nazionale Universitario di Calcolo Elettronico* (CNUCE, National University Center for Electronic Computing) to host the successor of the CEP, an IBM 7090 to offer computing services to the Italian scientific community at large. At the same time, a wealth of educational activities were carried on, from internal courses on the use of the CEP to solve Numerical Analysis problems in the '50s, to graduate courses in Computing in the early '60s, to lectures offered within Science and Engineering undergraduate courses in the second half of the '60s, and finally the first undergraduate course in *Scienze dell'Informazione* (Information Sciences) in 1969, as mentioned above, to finish with the first PhD program in Informatics in 1981. As a consequence, what is described below is just a very tiny part of the software that has been developed during the developments outlined above.

The expertise and the computing facilities available in Pisa as a consequence of this history fostered a large number of scientific endeavours involving the development of software. Many of these projects were, and are, internal to Informatics, but many others in Physics, Mathematics, Medicine, etc. required software. In the sequel, we report on some of what we were able to save so far, covering a representative range of situations with respect to the target of the software, but also to the acquisition mode, the development period, the programming language, the versioning system, etc.

All the software projects that have been acquired following the SWHAP guidelines are indexed in the SWHAPPE catalogue [13].



(a) Handwritten FORTRAN CEP form.



(b) FORTRAN CEP printout.

Figure 3: Excerpts of Softi raw materials.

A. *Softi (1968) [14]*

This code, though very short, is interesting for both its purpose and its structure. Its purpose is described as *Softening*[5] *of a curve* and performs a simple smoothing of the values of a function. Likely, it is an exercise in using the CEP, the implementation of a general purpose numerical algorithm. As such, it is representative of the subsequent activities of the author, Tonina Starita – at the time affiliated to the CSCE – who was pioneering the use of digital computing machines to study biological data, like EEG and similar measurements.

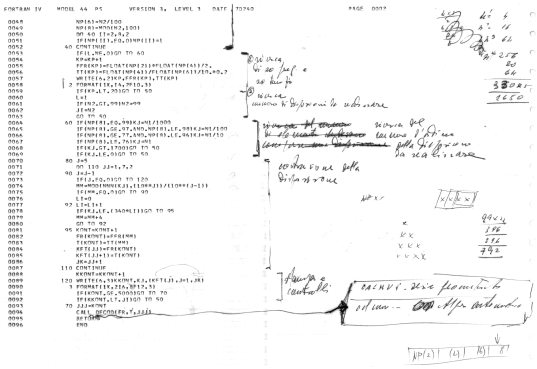
The structure of this code is interesting, since it is representative of the typical usage of the CEP at the time, and comprises three kinds of code. The main algorithm is written in FORTRAN CEP[15], a dialect, developed in–house at CSCE, of the IBM FORTRAN II; the input/output routines are written in the assembly language of the CEP (LSDC, Symbolic Decimal CEP Language); finally there are also the scripts to launch execution, written in the command language of the execution control system of the CEP.

The original materials were stored in the MSC's warehouse by an unknown person in a folder labeled *CEP Programs*, and recovered by Carlo Montangero in July 1919, while looking for source code documenting the early usage of the CEP. Besides Softi, the folder contains thirteen more manila folders, which seems to be all is left of the software run on the CEP.

The original Softi documents consist of several sheets of different formats and substance. There are the sheets hand written by Tonina with the original source code, and the related printouts, once the code had been punched on paper tape by the service personnel (Fig. 3). The former are written on printed forms, prepared by the Programming Service of the CSCE. The printouts have been cut from a continuous roll of paper, and show hand annotations of the servicing personnel, like "after assembly, the machine halted at 4095". The source code, given that it is very short, was digitized typing it directly in Visual Studio Code, and then imported in GitHub. The most interesting piece is the FORTRAN program, which occurs in four versions. Three are printed; one, version 3, was recovered from hand annotations on version 2.

The resulting repository consists of 56 lines of FORTRAN CEP code in 4 versions, 48 lines of CEP Assembly in a single version and 4 lines of CEP PDT in 2 versions, from 1968.

[5] Actually, the first version reads *Softing*, hence the name we chose for the repository.



(a) Annotated FORTRAN printout.

(b) TAUmus on exhibit.

Figure 4: TAUmus materials and showcase.

B. TAUmus (1972-1977) [16]

TAUmus is computer music software developed during the 70's, first at IEI and then at CNUCE, by a team led by Maestro Pietro Grossi. It was basically a command line interpreter running on the IBM 370 that had substituted the 7090 at CNUCE in the meantime, and it enabled the user to create and modify music to be played by the TAU2 audio terminal. This was a hardware device that was developed by the same team at IEI, to play, under the control of the 370, the music created via TAUmus.

The acquisition process started as an initiative of the SWHAPPE team during the aforementioned celebrations for the 50th anniversary of the first course in Computer science: TAUmus looked worth preserving as one of the first computer music software that has ever been developed. Furthermore, despite the small amount of source code that has been recovered, TAUmus is quite interesting for several reasons: indeed, the fact that this software has been only partially recovered (thus presenting "holes") pose interesting questions about the preservation of incomplete software. TAUmus also provided us a benchmark for the entire process, including the (optional) *present* phase. Indeed, thanks to a collaboration with ISTI-CNR[6], we have been able to provide a TAUmus demonstrator showcasing some of the tracks created by Grossi in the '70s (Fig. 4b).

After a first meeting with Leonello Tarabella, where the team has been provided with the available material (mostly FORTRAN/TAUmus listings and handwritten notes - see Fig. 4a), the team proceeded to digitize and archive the material in the Depository, while the source code has been typed into FORTRAN files in the TAUmus workbench.

The resulting repository consists of 266 lines of FORTRAN 77 code in 4 files in 2 versions from 1972. Furthermore, the repository contains 71 lines of TAUmus code in 4 files, contained in the TAUmus folder.

C. CMM (1994-1997) [17]

The Customizable Memory Management (CMM)[18] is a memory management facility supporting complex memory-intensive applications in C++. The CMM can manage several heaps, each one implementing a different storage discipline. It offers some predefined storage discipline (e.g. copying garbage collection and mark&sweep heap) allowing users to define their own heap classes for the specific storage requirements of their algorithms. The CCM has been exploited in the implementation of the Buchberger [19] algorithm in the context of the European research project PoSSo (Polynomial System Solving) active from 1992 to 1995, and in 1994 was used by Sun Microsystems in the development of the language the Oak programming language, later known as Java.

CMM was our first acquisition: After the seminar that he gave to present Software Heritage at the Scuola Normale in Pisa in December 2018, Di Cosmo met accidentally Attardi, who offered to retrieve the source code of the CMM. The offer was promptly accepted, and the code served as test bed during the development of SWHAP and its implementation in Pisa.

The raw material consists in a compressed tgz archive of eight (out of nine) versions managed with an in-house versioning system, along with build instructions. The need to synthesize a development history in git, prompted the implementation of a general purpose tool for reconstructing a Git repository from a directory of source code [20]. The tool is now available and used for the other acquisitions.

The resulting repository (Fig. 5) consists of 10k lines of mainly C++ code in 41 files in 8 versions from 1994 to

[6]This is the institute born within CNR from the fusion of IEI and CNUCE.

1997.

D. OrbFit (2002-2019) [21]

OrbFit is an astronomy library to compute orbits and ephemerides [22]. The first version of OrbFit dates back to 1995, although its development builds upon a pre-existing system, Orbit, which had already reached the eleventh version at the time.

Orbit can be dated back to the founding of the Celestial Mechanics Group (1978), but we have not yet found any trace of these older versions.

This acquisition allowed us to test yet another aspect of the SWHAP process. Indeed, within the Informatica50 initiatives mentioned above, the University issued a call to submit source code to Software Heritage, and the current developers promptly answered – a typical case of acquisition in *pull mode*.

It is worthwhile to note here that, though widely used and maintained, the current code was stored on a local server, with no special provisions to ensure its survivability, as it happens, too often, for lack of resources.

The older versions were rescued from the hard drives of decommissioned computers in the Department of Mathematics. The acquisition consisted in recovering the versions from the different digital media and making them versionable, that is, giving them a common structure. To collect metadata we have carried out some interviews.

The resulting repository consists of 78k lines of code in many programming languages (FORTRAN 77, FORTRAN 90, Perl, Tcl/Tk, Pascal, Prolog, D, Mercury, MATLAB) in more than 300 files in 13 versions from 2002 to 2019.

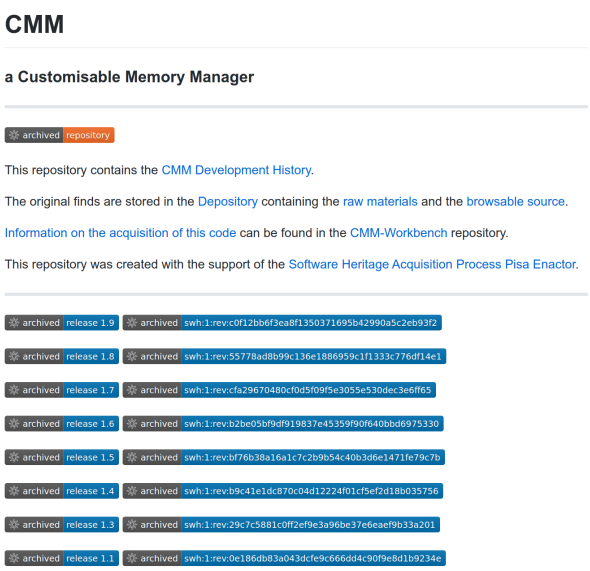


Figure 5: The front page of the CMM repository.

V. Conclusions

Software source code being a relatively young creation of human ingenuity, and until recently little known to the general public, it is not surprising that initiatives to preserve it have been way less widespread than for other digital material that encodes more familiar forms of human knowledge, like written text, music, images or videos.

Today, there is a raising awareness of the importance of preserving landmark legacy source code, as an important trace of humankind's efforts to build the science and the technology underlying the digital revolution.

There is also a clear urgency to act while the creators of these software artifacts are still alive, and available to provide precious insights about where to find their source code, when and why it was produced, and how it links to other important documentary evidence like research articles or documentation.

Since rescuing, curating and preserving landmark legacy source code is a significant undertaking that requires considerable human resources and time, it is of paramount importance that the result of the preservation effort be uniform, faithful and of good quality. And yet, to the best of our knowledge, up to now there were no comprehensive guidelines specifically for software source code.

The detailed guidelines collected in the SWHAP process have been developed in collaboration by Software Heritage, the University of Pisa and UNESCO precisely to fill this gap.

They are meant to ensure that when legacy source code is available, its history will be reconstructed into modern version control systems, and equipped with proper metadata that describes the work of the original authors, as well as the work of the curation team.

SWHAP has been extensively tested and refined while rescuing a variety of legacy source codes, ranging from very old small programs that reached us only in printed form, to large software projects that are still actively maintained today.

The selection presented in the previous section shows how all these diverse cases can be successfully handled using SWHAP, leading to uniform, high quality results.

We hope that these guidelines will empower a large community of digital preservationists, and that the concrete results summarised in this article will help attract more people willing to contribute to rescue the precious software source code legacy of the digital revolution.

References

[1] H. Abelson and G. J. S. with Julie Sussman, *Structure and Interpretation of Computer Programs*. Cambridge, MA: The MIT Press and McGraw-Hill, 1985,

```
163 force_model.f90:      END SUBROUTINE force
164 force_model.f90:END MODULE force_model
165 force_model.f90:LOGICAL FUNCTION velocity_req()
166 force_model.f90:END FUNCTION velocity_req
167 force_model.f90:SUBROUTINE radar_ob(type,m)
168 force_model.f90:END SUBROUTINE radar_ob
169 force_model.f90:SUBROUTINE selpert(name,found)
170 force_model.f90:END  SUBROUTINE selpert
171 force_model.f90:SUBROUTINE selpert2(nam0,namp,nfound)
172 force_model.f90:END SUBROUTINE selpert2
173 gaussdeg8.f90:SUBROUTINE gaussdeg8(tobs,alpha,delta,obscod,el,nroots,nsol,rr,fail,msg,debug)
174 gaussdeg8.f90:END SUBROUTINE gaussdeg8
175 least_squares.f90:! NOT IN MODULE
176 least_squares.f90:! MODULES AND HEADERS
177 least_squares.f90:MODULE least_squares
178 least_squares.f90:! PUBLIC SUBROUTINEs
179 least_squares.f90:! LIST OF PRIVATE ENTITIES, COMMON TO THE MODULE
```

```
184 force_model.f90:END SUBROUTINE selpert2
185 force_sat.f90:MODULE force_sat
186 force_sat.f90:CONTAINS
187 force_sat.f90:   SUBROUTINE forcesat(x,v,t0,f,nd,idc,xxpla,ips,imem, derf0)
188 force_sat.f90:  END SUBROUTINE forcesat
189 force_sat.f90:    SUBROUTINE eamoon_mass
190 force_sat.f90:    END SUBROUTINE eamoon_mass
191 force_sat.f90:END MODULE force_sat
192 gaussdeg8.f90:SUBROUTINE gaussdeg8(tobs,alpha,delta,obscod,ecc_max,q_max,el,nroots,nsol,rr,fail,msg,
193 gaussdeg8.f90:END SUBROUTINE gaussdeg8
194 laplace_poincare.f90:SUBROUTINE laplace_poincare(m,obs,obsw,el,nroots,nsol,rr,fail,msg,debug)
195 laplace_poincare.f90:END SUBROUTINE laplace_poincare
196 least_squares.f90:! NOT IN MODULE
197 least_squares.f90:! MODULES AND HEADERS
198 least_squares.f90:MODULE least_squares
199 least_squares.f90:! PUBLIC SUBROUTINEs
200 least_squares.f90:! LIST OF PRIVATE ENTITIES, COMMON TO THE MODULE
```

Figure 6: An excerpt of OrbFit synthetic development history: comparing two versions.

pp. xx + 542, isbn: 0-262-01077-1 (MIT Press), 0-07-000422-6 (McGraw-Hill).

[2] L. J. Shustek, "What should we collect to preserve the history of software?" *IEEE Annals of the History of Computing*, vol. 28, no. 4, pp. 110–112, 2006. doi: 10.1109/MAHC.2006.78. [Online]. Available: http://dx.doi.org/10.1109/MAHC.2006.78.

[3] E. G. Report, *Paris call: Software source code as heritage for sustainable development*, Available from https://unesdoc.unesco.org/ark:/48223/pf0000366715, 2019.

[4] J.-F. Abramatic, R. Di Cosmo, and S. Zacchiroli, "Building the universal archive of source code," *Commun. ACM*, vol. 61, no. 10, pp. 29–31, Sep. 2018, issn: 0001-0782. doi: 10.1145/3183558. [Online]. Available: http://doi.acm.org/10.1145/3183558.

[5] D. Spinellis, "A repository of unix history and evolution," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1372–1404, 2017. doi: 10.1007/s10664-016-9445-5. [Online]. Available: https://doi.org/10.1007/s10664-016-9445-5.

[6] R. Burkey, *Virtual agc - changelog*, Available at http://ibiblio.org/apollo/changes.html, Spans years 2003 to 2019.

[7] L. Bussi, R. Di Cosmo, C. Montangero, and G. Scatena, *Software heritage acquisition process guidelines*. [Online]. Available: https://github.com/SoftwareHeritage/swhapguide.

[8] G. A. Cignoni and F. Gadducci, "Retracing and assessing the CEP project," *CoRR*, vol. abs/1904.00944, 2019. arXiv: 1904.00944. [Online]. Available: http://arxiv.org/abs/1904.00944.

[9] K. Cox, P. Aufderheide, P. Jaszi, and B. Butler, *Code of best practices in fair use for software preservation*, Sep. 2018. [Online]. Available: https://www.softwarepreservationnetwork.org/project/code-of-best-practices-for-fair-use/.

[10] *First steps to contribute to wikipedia*. [Online]. Available: https://commons.wikimedia.org/wiki/Commons:First_steps/Contributing.

[11] *Donating data to wikidata*. [Online]. Available: https://www.wikidata.org/wiki/Wikidata:Data_donation.

[12] G. community, *Git version control system*, retrieved 09 April 2018, 2005. [Online]. Available: https://git-scm.com/.

[13] *Software heritage acquisition process pisa enactor*. [Online]. Available: https://github.com/Unipisa/SWHAPPE.

[14] *Software Heritage depository: Softi - fortran cep code to smooth a curve.* [Online]. Available: https://archive.softwareheritage.org/browse/origin/https://github.com/Unipisa/Softi.git/directory/.

[15] O. G. Mancino and M. Morandi Cecchi, "The internal structure of the fortran cep translator," *Comm. ACM*, pp. 149–151, 1965. doi: 10.1145/363791.363799.

[16] *Software Heritage depository: TAUmus - code controlling TAU2, a music synthesizer of the 70's.* [Online]. Available: https://archive.softwareheritage.org/browse/origin/https://github.com/Unipisa/TAUmus/directory/.

[17] *Software Heritage depository: CMM - a customisable memory manager.* [Online]. Available: https://archive.softwareheritage.org/browse/origin/https://github.com/Unipisa/CMM/directory/.

[18] G. Attardi and T. Flagella, "Memory management in the posso solver," *J. Symb. Comput.*, vol. 21, no. 3, pp. 293–311, 1996. doi: 10.1006/jsco.1996.0013. [Online]. Available: https://doi.org/10.1006/jsco.1996.0013.

[19] B. Buchberger, "A theoretical basis for the reduction of polynomials to canonical forms," *SIGSAM Bull.*, vol. 10, no. 3, 19–29, Aug. 1976, issn: 0163-5824. doi: 10.1145/1088216.1088219. [Online]. Available: https://doi.org/10.1145/1088216.1088219.

[20] *Directory tree to synthetic git*. [Online]. Available: https://github.com/Unipisa/DT2SG.

[21] *Software Heritage depository: OrbFit - an astronomy library to compute orbits and ephemerides.* [Online]. Available: https://archive.softwareheritage.org/browse/origin/https://github.com/Unipisa/OrbFit.git/directory/.

[22] A. Milani and G. Gronchi, *Theory of Orbit Determination*. Cambridge University Press, 2009. doi: 10.1017/CBO9781139175371.