United Nations
Educational, Scientific and
Cultural Organization

Memory of
the World

Università di Pisa

inventors for the digital world

The Software Heritage Acquisition Process

Software Heritage
THE GREAT LIBRARY OF SOURCE CODE

TABLE OF CONTENTS

Laura Bussi
Dept. of Computer Science, University of Pisa
l.bussi1@studenti.unipi.it

Roberto Di Cosmo
Software Heritage, Inria and University of Paris
roberto@dicosmo.org

Carlo Montangero
Dept. of Computer Science, University of Pisa
carlo@montangero.eu

Guido Scatena
Dept. of Computer Science, University of Pisa
guido.scatena@unipi.it

Project institutional contacts

University of Pisa:
Chiara Bodei, Fabio Gadducci

UNESCO:
Davide Storti

Inria/Software Heritage:
Roberto Di Cosmo

CLD 1658.19 CI-2019/WS/8

FOREWORD

Moez Chakchouk,
Assistant Director-General for Communication and Information, UNESCO

Documentary heritage comprises analogue documents or digital informational content of "significant and enduring value to a community, a culture, a country or to humanity generally, and whose deterioration or loss would be a harmful impoverishment. Significance of this heritage may become clear only with the passage of time", states the UNESCO Recommendation concerning the Preservation of, and Access to Documentary Heritage, including in Digital Form (UNESCO, 2015).

Following the opening of the world's first global archive of software source code in 2018, the Paris Call on Software Source Code as Heritage for Sustainable Development (UNESCO, 2019) has established a firm basis for the recognition of software source code as a receptacle and expression of part of our common knowledge. Preservation of documentary heritage, however, is an ongoing process requiring clear and agreed upon techniques, treatments and procedures, making the knowledge embodied in software source code accessible and reusable. In this respect, the Software Heritage Acquisition Process represents a unique contribution by the University of Pisa and, with Inria, a remarkable example of international cooperation as envisioned by the signatories of the Paris Call.

In this spirit, UNESCO encourages software developers, memory institutions, the business sector, academia and civil society as a whole to support these efforts, fostering international cooperation to build a common framework for software preservation and access.

Paolo Mancarella,
Rector, University of Pisa, Italy

The software is everywhere; it is the framer of our present. It is part of our cultural heritage, although not perceived as such because we take it for granted as the land on which we stand. Despite its power, however, the software is a fragile tool, doomed to disappear, unless saved and adequately preserved. This is why, the University of Pisa is pleased and honoured to support the Software Heritage Project, proposed by Inria in partnership with UNESCO.

Becoming part of the project network and giving our contribution is a constructive way to celebrate the history of information technology and the role our University played, as we are celebrating the anniversary of the first degree in Computer Science in Italy, launched fifty years ago, here in Pisa.

However, our link with information technologies goes back much further, as during the fifties our University was responsible for the construction of the first scientific mainframe designed in our country. We, therefore, believe that joining this project is a duty, rather than only a pleasure: it is necessary to preserve the historical software built during the past half-century.

Moreover, it is significant that the first important step in this collaboration is the development of the Software Heritage Acquisition Process: a tool and a methodology created to collect and preserve the software of historical, scientific and cultural relevance. The preserved materials are intended to become a real roadmap for current and future information technology historians, thus enabling us to leave the doors wide open to the virtual worlds of our present.

Bruno Sportisse,
CEO, Inria, France

For 50 years Inria has been at the forefront of research and innovation in the digital sciences that were at the origin of the digital revolution. Software has played a central role in the work of Inria's researchers which have produced over the past decades several thousand software packages, some of which have led to major scientific breakthroughs and others that formed the basis of successful commercial products in one of the many Inria spin-offs. At the heart of software, there is the source code, a special form of knowledge that is both readable by humans, who write and evolve it, and directly translatable into a form executable by a machine.

We believe that the source code of the software developed since the beginning of the computing era, a little more than half a century ago, constitutes a precious cultural heritage: it is important to collect it, preserve it and make it available to everyone. That's why Inria initiated Software Heritage, in partnership with UNESCO, and is fully committed to foster the emergence of an international network of entities that share this vision.

Today we are delighted to see one more step forward, with the publication of the Software Heritage Acquisition Process developed together with the University of Pisa: it is an important contribution to the ongoing efforts and provides detailed guidelines for those that want to embark in the exciting journey of rescuing, curating and archiving landmark legacy source code.

Roberto Di Cosmo,
CEO, Software Heritage

Software is at the heart of our digital society and embodies a growing part of our scientific, technical and organisational knowledge. The core mission of Software Heritage is to collect, organise and preserve the source code of all this software, which constitutes a precious part of our own cultural heritage, and pass it over to future generations.

We do this for multiple reasons. To preserve the knowledge embedded in software source code. To allow better software development and reuse for society and industry. To foster better science, building the infrastructure for preserving, sharing and referencing research software, a stepping stone for reproducibility, and a necessary complement to Open Access.

We do this now, because we are at a turning point: most pioneers of the digital age are still around, and willing to contribute their knowledge, but only for a limited time. They are spread all around the world, and the amount of knowledge at risk of being lost is huge, so we'll only succeed if a broad international community steps up to tackle this noble task.

The Software Heritage Acquisition Process described in this document is the result of long months of intense work in collaboration with the University of Pisa, with the goal of jumpstarting this effort by empowering all those that are interested to contribute. It provides concrete, actionable guidelines for properly rescuing and curating legacy landmark source code. We hope to see it broadly adopted, for the benefit of society as a whole.

INTRODUCTION

Software is everywhere, binding our personal and social lives, embodying 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, a rapidly increasing part of our collective knowledge is embodied in, or depends on software artifacts.

Software does not come out of the blue: it 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" [1], and "provides a view into the mind of the designer" [2].

As stated in the Paris Call on Software Source code as Heritage for sustainable development [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 [4].

For software that is easily accessible online, and that can be copied without specific legal authorizations, the approach is based on automation. This way, as of September 2019, 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 [5] and the source code of the Apollo Guidance Computer [6].

This document presents the first version of SWHAP, the ***SoftWare Heritage Acquisition Process*** 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.

The next section provides an *abstract* view of SWHAP, its steps, documents and resources. No specific assumptions on the *tools, platforms and technologies* that may be used to enact it are made, but some requirements are made explicit.

The last section describes how the abstract process is implemented at the University of Pisa by leveraging the Git toolset and the GitHub collaborative development platform. This implementation is named SWHAPPE (SWH Acquisition Process Pisa Enactor) in this document.

Resources available at https://www.softwareheritage.org/swhap complement this document. This includes an annotated example, using a real world medium-sized software project [7], as well as a list of tools that may be helpful for other landmark legacy software source code rescue teams.

Revised versions of this document will be published as needed.

www.softwareheritage.org/swhap

THE PROCESS, ABSTRACT VIEW

This section describes SWHAP, 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.

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.

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 CEP, the first Italian computer) 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, and 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.

Curate

The purpose of this phase is *to analyze, cleanup and structure* the raw material that has 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 using 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].

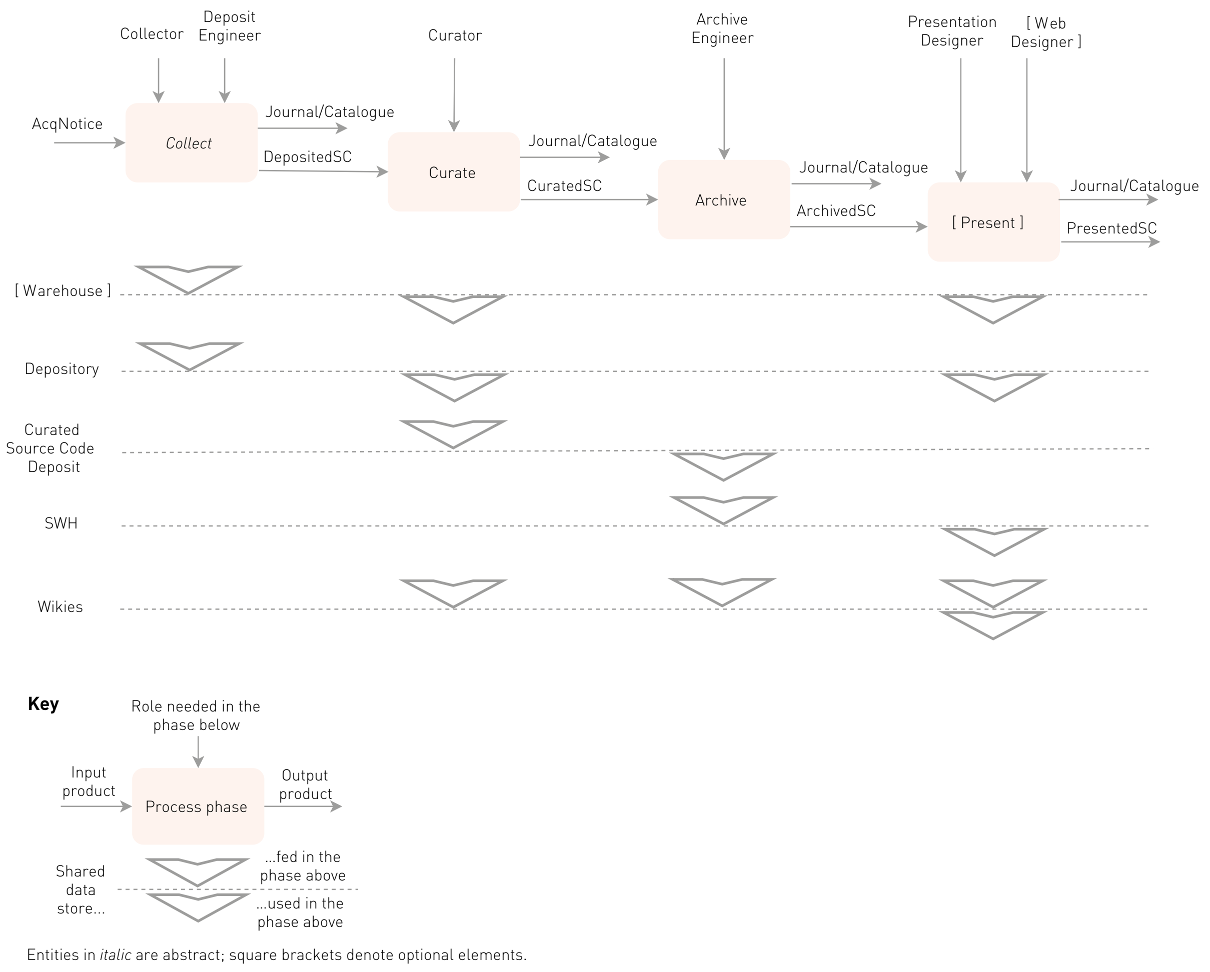
Archive

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 https://commons.wikimedia.org/wiki/Commons:First_steps/Contributing) and Wikidata (see https://www.wikidata.org/wiki/Wikidata:Data_donation), hence we will focus primarily on curating and contributing the software source code to Software Heritage, a process that is new and may require rather technical steps.

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 https://www.softwarepreservationnetwork.org/bp-fair-use/

Figure 1. Source code acquisition process.



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

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.

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.

Warehouse

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

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.

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.

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.

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.

Collector

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

Deposit engineer

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

Curator

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

Archive engineer

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

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.

Remark the technical resources described above in abstract terms, may be implemented in a variety of ways. For example, one can imagine a single Depository for all the software projects that are collected, but it is also possible to use a separate Depository for each software project, and the same holds for all the other areas.

Remark the roles indicated above need not necessarily be played by different persons, e.g., Collector and Curator may be the same person, nor be played by a unique person, e.g., there can be several cooperating Curators, in case of large systems.

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

Implementation requirements

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

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).

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.

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.

Openness

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

Interoperability

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

THE ACQUISITION PROCESS, A CONCRETE VIEW

In order to implement SWHAP, the first step is to decide how to instantiate the needed storage and working areas: Warehouse, Depository, Curated source code deposit and Workbench.

The Warehouse is quite similar to the usual storage area where museums preserve their collections; it will need to be set up in a specific physical location, following the well-established process for museums, so we will not cover it in this guide.

The other areas, which are virtual spaces, can very well be set up using distinct digital platforms, but it is also possible to instantiate all of them on a single platform.

This choice was made for the SWHAP Pisa Enactor (SWHAPPE), the implementation adopted by the SWHAP@Pisa project: SWHAPPE exploits the collaborative platform GitHub (https://github.com/) as a host platform for all the virtual support areas of the process.

The solutions adopted in SWHAPPE are described in detail in this section, together with their rationale.

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

General Motivation for using Git and GitHub

The choice of Git as the designated tool for traceability and historical accuracy, and of GitHub as the unifying platform to support the SWHAP process proceeds from several considerations that we review below.

First of all we discuss the choice of *Git*. One of the key requirements set forth for SWHAP is the need to ensure *full traceability* of the operations performed on the recovered digital assets. This means that each of the virtual places must provide means to record the history of the modifications made to the digital assets, with information on *who did what and when*. It is very convenient to use the same tool in all of the virtual places of the process, as this reduces the learning effort and streamlines the process. All modern version control systems provide the needed functionality, and we have chosen *Git* as our standard tool, as it is open source (another of our requirements) and broadly adopted. *Git* is a powerful tool, and requires some expertise to make the most out of it. However, a large part of the process is scriptable, and this will hide the underlying complexity to the final user, which can then focus on the main issue: curating and preserving the code and its history.

Another important motivation for our choice of Git is the ability to support *historical accuracy*, i.e., providing a faithful view of the history of both the recovered source code and the acquisition process, as prescribed by the SWHAP key requirements. This is properly accommodated by the commit and versioning mechanisms offered by Git, which allow separating authors from committers: this way on can record both the story of the original software and the story of its curation.

Finally, we had to choose one of the many online platforms that allow to collaborate using *Git*. GitHub, GitLab.com and Bitbucket are the most known ones and are all regularly archived in Software Heritage, so that *long term availability* of their contents is preserved, no matter which one of these platforms is chosen.

Among all these platforms, GitHub is by far the most popular and active, and is also the platform adopted by the University of Pisa, so it was a natural choice, and we believe this will make the learning curve gentler for most SWHAP adopters.

In the following, we provide detailed guidelines to instantiate the process using Git on GitHub. We think that most of what is described in the guide can be easily adapted to any of the other *Git*-based collaborative platforms.

SWHAP - GitHub correspondence

SWHAPPE is a straightforward implementation of the abstract process, which concretizes the (logical) areas described above by means of *repositories* in GitHub: there are three repositories for each source code acquisition, one for each area of the abstract process:

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.

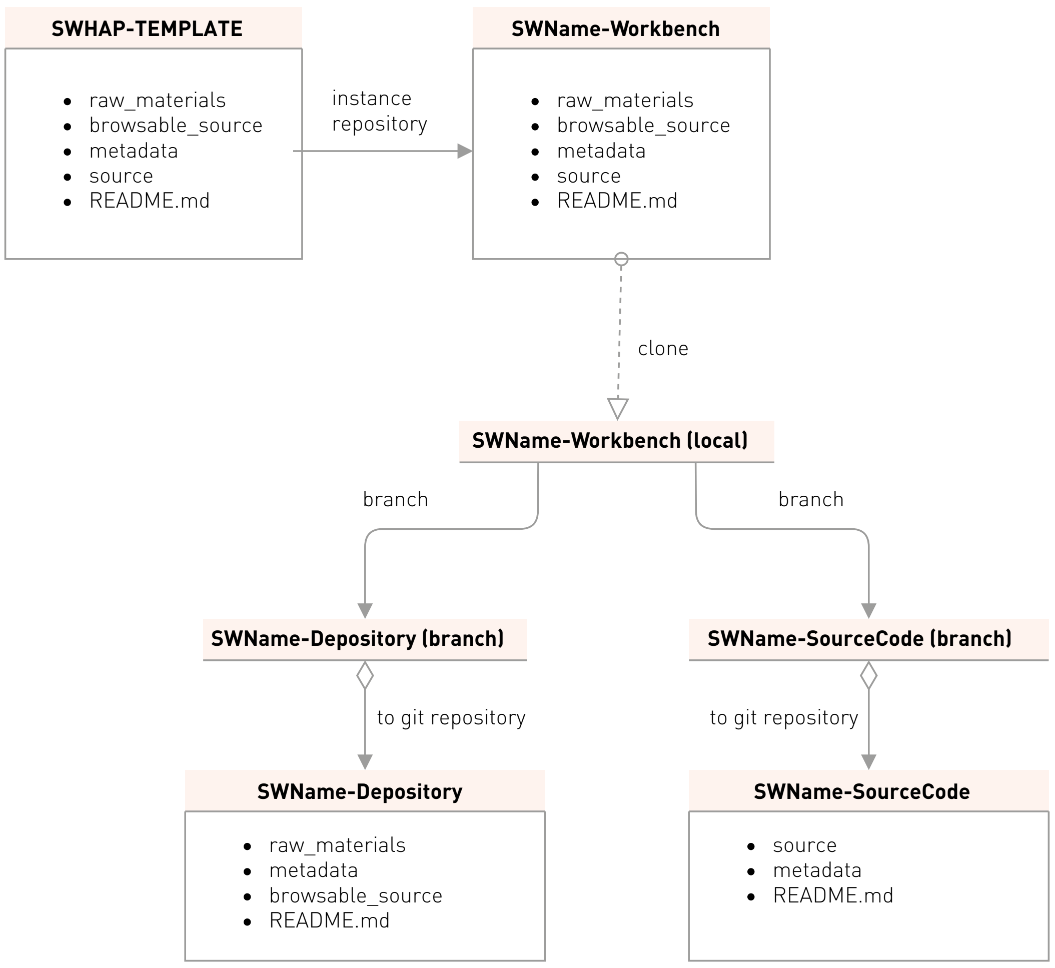
Let's remark that SWHAPPE has *different* Workbench and Depository repositories for each code acquisition, but it would also be possible to use a single Workbench repository and/or a single Depository repository to work on all the collected software, provided one maintains a well-organised directory structure which keeps the codes separated. On the other hand, we *need* a Source Code repository for each software project, to be actually ingested in the Software Heritage archive.

Process overview

GitHub features *template* repositories that can be instantiated whenever needed (see https://help.github.com/en/articles/creating-a-template-repository). We used this feature in SWHAPPE, and designed a repository, SWHAP-TEMPLATE, that embodies the core support to enact the process. Its structure and use is shown in figure 2. In the picture and in the following *SWName* is a variable that takes the name of the acquired code as its value at each instantiation.

Once SWHAP-TEMPLATE has been instantiated, the *SWName-Workbench* repository so created need to be

Figure 2. Overview of the SWHAPPE process.



cloned to the user's machine, so that he can work on the collected files locally - the Git clone mechanism ensures that these changes can be safely moved to the original repository, for publication and sharing with other actors in the acquisition.

We create two dedicated *branches*[4], that allow to track separately the operations that will be later moved to the Depository and the Development History Deposit: *Depository*, to contain the raw materials and the browsable sources as well as the metadata, and *SourceCode* to organize the source code in view of the reconstruction of its development history. Finally, the *Depository* and *SourceCode* branches become two repositories: the latter is shipped to the Software Heritage archive, the former is published by the organization promoting the acquisition.

4 For more information on branches, see https://help.github.com/en/articles/github-glossary

The SWHAP template

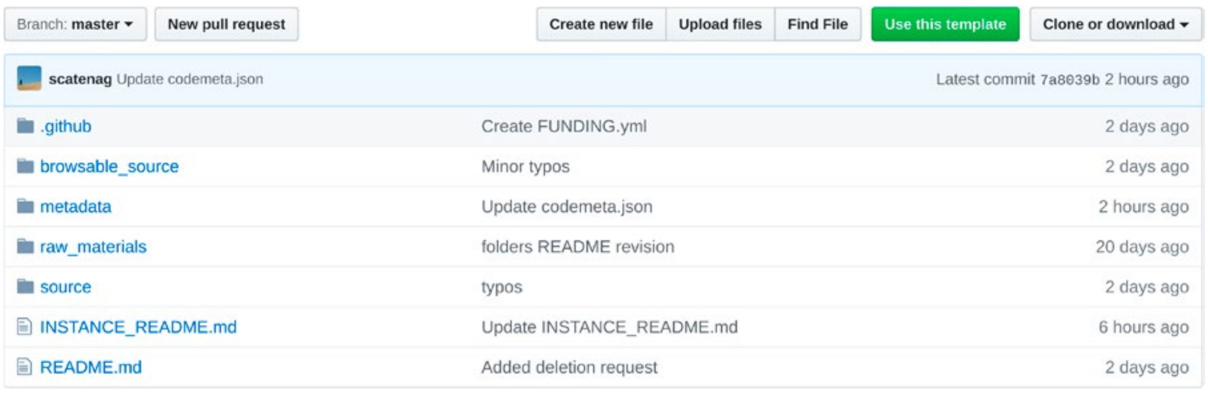
The structure of the template is shown in fig. 3.

First of all, we can see a correspondence between the Depository presented in the process and the area provided by `raw_materials` and `browsable_source`: indeed, these two folders will be moved in order to instantiate the Depository, once they have been loaded, the former with the original materials, just as they have been found or submitted, the latter with a first revision of the source code, made accessible through the GitHub web interface, e.g., archives should be decompressed, code transcribed from pictures, etc.

The `source` folder is provided as the starting point for the creation of the Source Code *Git* repository, in the curation phase. The curator has to recognize each major version of the code, and refactor it accordingly - one separate folder per each version. To create the Source Code Deposit, however, we exploit the *commit* and *versioning* mechanisms of *Git*: more on this later on.

As for the `metadata` folder, here we record all the information about the software and the acquisition process (catalogue, actors, journal, etc.). The guidelines to fill this part are given in the template itself.

Figure 3. Top structure of the Template repository.



scatenag Update codemeta.json — Latest commit 7a8039b 2 hours ago

.github	Create FUNDING.yml	2 days ago
browsable_source	Minor typos	2 days ago
metadata	Update codemeta.json	2 hours ago
raw_materials	folders README revision	20 days ago
source	typos	2 days ago
INSTANCE_README.md	Update INSTANCE_README.md	6 hours ago
README.md	Added deletion request	2 days ago

The process, step by step

Instantiation

The first step is to create an instance of the SWHAP-TEMPLATE[5], that should be named SWName-Workbench, and then to clone it to obtain a local copy on your machine[6].

From this point on, you'll be able to upload files and to modify/copy/move them locally, and use *Git* commands to push changes to GitHub.

Let us now see the steps to be followed, together with some explanations.

Collect phase

Upload files in `raw_materials`

All the collected files must be uploaded in the `raw_materials` folder.

If there are physical materials, folder `raw_materials` should contain a reference to the related Warehouse, that may follow the Spectrum guidelines [8].

Move the source code to `browsable_source`

All the source code files must then be put into the `browsable_source` folder.

If the raw material is an archive, you should unpack it locally and then upload the result on GitHub by performing a push[7].

5 See the documentation on https://help.github.com/en/articles/creating-a-repository-from-a-template

6 See the documentation on https://help.github.com/en/articles/cloning-a-repository

7 See the documentation on https://help.github.com/en/articles/adding-a-file-to-a-repository-using-the-command-line.

If the code was only available in non-digital form (e.g. printed listings), you can either transcribe it manually, or use a scanner and an OCR (optical character recognition) tool to parse it. See Appendix A for a list of suggested tools.

Particular care should be used to ensure the files in `browsable_source` have the correct extension: scanner and OCR usually generate files with a generic `.txt` extension that must be changed to the extension typically used for the programming language they contain.

Note that, at this stage, we are not interested in precise information about the versions of the software. The purpose is to have machine-readable documents.

Finally, in preparation for the curation phase, you may want to copy the files in `browsable_source` to the `source` folder.

Create Depository

The next step is to create the branch Depository, containing only the folders `raw_materials` and `browsable_source`, together with the metadata updated to this point. Then, create the Depository repository from this branch.

Curate phase

Curate the source code

Once the Depository creation is complete, you can move back to the `source` folder in the master branch. Here you have to divide and number the versions, putting the files of each one in a dedicated folder and determining who did what and when.

In practice, this means that *for each version of the software* you need to ascertain:

- the *main contributing author*,
- the *exact date* of the release of this particular version

This information should be consigned in a dedicated metadata file, `version_history.csv`, having the fields described in Figure 4.

Figure 4. Fields of the version_history.csv file

`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*

(Re-)Create the Development History

Now we are ready to (re-)create the development history of the software. First you need to create a branch Source Code, with the *src* folder.

Then, you can proceed in two ways:

- *manually*: using the *Git* commands to push the successive versions into the `source` folder, reading the information collected in the file `version_history.csv` to set the fields for each version to the values determined during the curation phase;
- *automatically*: using a tool that reads the information from `version_history.csv` and produces the synthetic history in a single run; one such tool has been developed, DT2SG (https://github.com/Unipisa/DT2SG) , and you can see a running example in the full document available at www.softwareheritage.org/swhap.

The result will be a branch that materializes the development history of the software via Git commits and releases.

Create the final repository

Finally you can create the "official" software repository, taking the versions history from the src branch and the metadata from the master branch.

Iteration

New material may be discovered after the process has been completed, triggering an iteration of some of the phases described above. In this case, we recommend proceeding as follows:

- if new raw material (non-source code) is found, we have to clone the Depository repository and add new items to it. In this way, the performed commits will correctly follow the previous ones.
- if new source code is found, after we collected it in the Depository, we have the following cases:

  (1) The recovered source code is related to a version, which is already included in the software history.

  (2) The source code represents a completely new version, with respect to the sw history as it was previously collected.

We are not finished yet, since in both cases the SourceCode repository is no longer consistent with the collected source code, and we have to recreate it, performing the following steps:

- Delete the SourceCode repository.
- Move back to the Workbench and according to the current case:

  if (1), add the source code to the correct version.

  if (2), add the new version folder with the related metadata.
- Recreate the software history as for the first iteration.

APPENDIX A • TOOLS THAT CAN HELP

Here is a list of tools for code acquisition and curation that have been used during the initial experimentation of SWHAPPE:

Used/suggested OCR:

- Tesseract (https://github.com/tesseract-ocr/). It can be installed and used from command line. An API is also provided to use the OCR in a script.
- OCR.space (https://ocr.space/). Online OCR and free API.

Dedicated scripts:

- DT2SG-Directory Tree 2 Synthetic Git (https://github.com/Unipisa/SWHAP-DT2SG). Creates the synthetic history of the software.
- SWHAP-EXAMPLE (https://github.com/Unipisa/SWHAP-EXAMPLE)

Many other tools exist, and are currently under construction and will be loaded on the SWHAPPE repository on GitHub.

ACKNOWLEDGMENTS

L. Bussi wants to acknowledge the Software Heritage Foundation for the scholarship that supported her work and the Department of Computer Science of the University of Pisa for hosting her while working on SWHAPPE.

BIBLIOGRAPHY

[1] H. Abelson and G. J. S. with J. Sussman, Structure and Interpretation of Computer Programs. The MIT Press: The MIT Press, 1985.

[2] L. J. Shustek, "What Should We Collect to Preserve the History of Software?," IEEE Ann. Hist. Comput., vol. 28, no. 4, pp. 110–112, 2006.

[3] Institut national de recherche en informatique et en automatique, Paris Call: Software Source Code as Heritage for Sustainable Development. UNESCO, 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.

[5] D. Spinellis, "A repository of Unix history and evolution," Empir. Softw. Eng., vol. 22, no. 3, pp. 1372–1404, 2017.

[6] R. Burkey, "Virtual AGC - Changelog," 2019. [Online]. Available: http://ibiblio.org/apollo/changes.html. [Accessed: 24-Sep-2019].

[7] G. Attardi and T. Flagella, "Memory Management in the PoSSo Solver," J Symb Comput, vol. 21, no. 3, pp. 293–311, 1996.

[8] T. Collections, "Introduction to Spectrum 5.0." [Online]. Available: https://collectionstrust.org.uk/spectrum/spectrum-5/. [Accessed: 24-Sep-2019].

The Software Heritage Acquisition Process

"Software is the fruit of a precious human artifact, the source code. Since our activities are more and more software-based, the code not only embeds the computational thought of its designer but it also has a lot to tell on our lives and our times. It is clearly part of our cultural heritage and it is essential to preserve it."

United Nations Educational, Scientific and Cultural Organization

Memory of the World

Università di Pisa

inventors for the digital world