Interdisciplinary Discussions about the Conservation of Software-Based Art

Community of Practice on Software-Based Art

A report written by PERICLES Project partners in collaboration with participants of the Community of Practice (CoP) on Software-Based Art.

A report by: Annet Dekker (Tate), Patricia Falcao (Tate)

From discussions with:
Agathe Jarczyk (Hochschule der Künste Bern), Arnaud Obermann (Staatsgalerie Stuttgart), Ben Fino-Radin (MoMA), Deena Engel (NYU), Dragan Espenschied (Rhizome), Emanuel Lorrain (Packed), Gaby Wijers (Lima), Joanna Phillips (Guggenheim), Katarina Haage (Tate), Kate Lewis (MoMA), Klaus Rechert (University of Freiburg/bwFLA), Mark Hellar (Hellar Studios LLC / SFMOMA), Martina Haidvogl (SFMOMA), Morgane Stricot (ZKM-Centre for Art and Media Karlsruhe), and Tom Ensom (Tate/ Kings College London)
Final edit by: Esther Harris (Tate)

Table of Contents

Introduction and Aims

In 2015/16 Tate organised a series of virtual discussions related to the preservation of software-based art as part of Tate's partnership in PERICLES, a European-funded project which focuses on the long-term digital conservation and preservation of digital resources, with particular focus on actively managing change and risk as part of this process. The idea for this series arose from the realisation that managing technical change in software-based art is not only a common concern for practitioners working in the field but also of interest to the research community. A group of engaged expert practitioners and researchers were invited to consider a set of topics at the core of the conservation of software-based artworks. Six discussion sessions were organised over a period of one year.

This report summarises the outcomes of these meetings and examines some of the key points.

The discussions were informed by a series of questions that we posed based on our collective experience of working with software-based artworks, as well as a survey of related studies in the conservation of new media, complex installations and software, in relation to art, digital forensics and archiving (see appendix). One of the main goals was to move beyond specific approaches or case studies and work towards more general statements about the conservation of software-based art. To achieve this, we organised the discussions around the following topics:

1. **Analysis and classification:** How do we currently analyse software-based artworks and identify risks for conservation and display? How might we develop our protocols? Can we develop general basic guidelines for this assessment? What do they need to include?
2. **Current preservation techniques:** Disk imaging, emulation and virtualisation and re-coding. What are the roles of artist, developer and conservator in the preservation process?
3. **Documentation and reporting:** How do we, and how might we in the future describe or model a digital environment? How do we currently capture and report on change within (existing) management systems or preservation workflows? How might we develop and improve these workflows?
4. **Quality control:** Once a conservation approach is chosen (for example to recode, virtualise or emulate) and executed, how can the 'new' version be compared and assessed against the 'original'? What is the status of the original source code and its hardware dependencies?

The focus group consisted of people from diverse backgrounds (ranging from computer science to time-based media conservation, digital conservation, and curation), working with different collections (in some cases without collections) and in different organisations (from large international and mid-size contemporary art museums, to universities and small specialised organisations). This variety was apparent throughout the discussions, as the experience and views expressed were directly related to the institutions and collections that people were involved with. For example, those in institutions engaged with software-based installation art felt the need to understand the individual details of each work. In this situation conservators will usually have a fairly small number of works and will have elements and instructions supplied by the artist. On the other hand, the concerns of those working with web-based or CD-ROM art were more focused on understanding a wider range of technical environments that would support as many works as possible. This reflects the fact that the collections mentioned (Rhizome and Transmediale) consist of hundreds of works, and also that these type of works were meant to be seen on most computers of a specific period in history. This report reflects these different approaches and acknowledges that each will benefit from understanding the other and where possible adopting common strategies.

The other key goal of this Community of Practice was to establish an international network of professionals with different backgrounds to share knowledge and develop practice. This was achieved, as evidenced by this report and the other outcomes of the project.

Discussion Outcomes

Classification and Terminology

There are many ways of looking at and trying to understand the new forms of art that have emerged alongside the rapid development of digital technologies. The metaphors used for understanding new media art depend on the cultural history professionals come from (Cook and Graham 2010:1) and also the purpose served by the classification and terminology. For example, conservation practitioners might find it useful to group works that share technical features relevant to particular preservation challenges, whereas a curator or art historian may wish to draw different types of connections, for instance around different types of visitor interaction.

This report focuses on discussions about classification and terminology as they relate to the current and developing conservation practice.

One of the main points of consensus was that a system of categorisation for software-based artworks is complex to both develop and use, since these works cover such a broad range of technologies and platforms. The complexities in and of the artworks themselves relate to the diversity of software and hardware being used in different ways, from anything that can be shown online to specific code used by artists for installation in specific physical settings. They are also related to the fact that different parts of a work can belong to multiple categories. In short we concluded that a classification would need to:

- Offer flexibility to accommodate diverse artworks
- Cover a broad range of technologies and platforms
- Change over time, through further reflection, research, learning and scholarship

These factors bring into question the viability or usefulness of having a single classification.

For the purpose of emulating software-based artworks it may be useful to consider a classification based on technical dependencies, for instance specific operating systems or interfaces. This type of classification can only exist alongside object-based description, and the understanding of how a work must be presented. Moreover, it was suggested that it is essential to come to a basic understanding of how computers work – the basic building blocks of computation – which only then could lead to a useful technical classification. When trying to conceptualise these building blocks it is helpful to find use-cases that move beyond a single artwork, to find abstractions that are useful and applicable.

For now, two generic descriptions of software-based art were seen as useful for conservation, as they point to the presence or absence of web-related dependencies:

- **Contained** software-based art: In which there is no external data input on the software
- **Networked** software-based art: In which the functioning of the work depends on data outside the control of collecting institutions

With regards to technical terminology, we wanted to clarify the meaning and use of the terms *emulation* and *performance*. In the past the term emulation was used by the Variable Media Network to mean re-creating an artwork by different means; in the context of art conservation it is often still used in this sense. However, we agreed that it would be more precise to use the

term emulation in the manner it is used in an engineering context, which refers to the recreation of hardware environments by means of software. In this report, emulation is therefore used in this narrower sense, and we recommend using different terms when referring to other techniques, for instance re-coding.

The term performance can be used for computer performance, i.e. how well the computer is doing what it is supposed to do; the performance of the software, i.e. how well the software is doing what it is supposed to do; or, in the sense that a software-based artwork is a performance. It is therefore important to clearly signpost the type of performance involved; it is no less important that artist, conservator and programmer all agree the ways in which these terms are used at the outset of a conservation process.

Role of the conservator, programmer and artist

The value of working with a programmer to help analyse and understand a software-based artwork is very clear to all the participants. There is also clear understanding and acknowledgement that at some point in time the code or software of an artwork may need to be changed. Eventually artists will not be present to approve specific changes, and therefore it is always best practice to ask the artist about their attitudes to change and what is important to preserve about a work. Moreover, dialogue between the conservator and the artist is likely to be ongoing over a number of years as at different moments decisions will need to be made in order to keep the piece running. This is no different from the kind of stewardship that is provided with other time-based artworks, where in the early years of a work and during the first iterations of the piece, conservators work closely with the artist to develop their knowledge and experience of the work and its behaviour, and how it may or may not change under different circumstances, different conditions, different curatorial preferences or against a changing technological landscape. For software-based artworks these iterations may not only be triggered by different exhibitions but also by risks associated with obsolescence or discontinuation of support of hard- and software.

To gain a better understanding of the identity of the work and its inherent vulnerabilities, collaboration between different disciplines is essential, i.e. the conservator may work closely with programmers or computer scientists. Based on this dialogue the preservation risks inherent in a work are identified, the degree and impact of that risk is estimated and mitigation or treatment options are considered and implemented. This is an engaged and ongoing learning

process, a collaboration over an extended period of time where all parties involved learn about the change, or the type of changes that the artist finds acceptable or not. Maintaining this thread of discussions is crucial for documenting and tracing key decisions.

Similar to other conservation practices, the conservation of software-based art is an educational process that needs to be lobbied for within an institution. To facilitate this, as has been happening for some time in many museums, curators and other staff members need to be invited to the conservation space to discuss challenges together in order to create more awareness across departments. In finding people to work with, conservators can approach their institution's IT department or staff member, who might be able to collaborate in finding solutions, or help find other people from their network to assist.

Source code

In computing, source code is described as "any collection of computer instructions, possibly with comments, written using a human-readable programming language, usually as ordinary text."[1] In museum practice there has been a great emphasis around preserving source code. Through our discussions and experience with different types of works and technologies, it became clear that this approach needs to be nuanced, as the value and usefulness of source code varies greatly depending on the artwork and the type of software. There was a lot of discussion about whether it was useful to collect the source code along with a work, or whether it might in some cases give collectors a false sense of security, or take up too many resources, possibly without enough returns. For fairly simple applications, for instance the work *Colors* by Cory Arcangel[2], source code may be used to recompile an artwork's software, as long as the chain of programming tools (or toolchain)[3] used for production is also available. This is not necessarily a simple task. Ideally, if an artwork is acquired and accessing the source code is relevant, then the whole chain of programming tools should also be collected to maintain access to that code. In many cases the source code can be unobtainable, or useless, for instance when a work exists within a virtual world such as Minecraft[4], or was created using gaming engines like Unity[5].

[1] https://en.wikipedia.org/wiki/Source_code
[2] http://www.coryarcangel.com/things-i-made/2009-054-colors-personal-edition
[3] A tool chain is the set of programming tools that is used to perform a complex software development task or to create a software product, which is typically another computer program or a set of related programs. For more information see: https://en.wikipedia.org/wiki/Toolchain
[4] https://minecraft.net
[5] https://unity3d.com

Besides its value for preservation, the other aspect discussed was what can be learnt from source code about the creation of a work, how it functions and its technical history. Source code analysis, as discussed by Engel and Wharton[6], can in some respects be considered akin to the chemical composition of a painting, to use Ben Fino-Radin's metaphor. The choice of language and platform, the coding style and structure, or the comments in the code are often reflections on an artist's practice and interests. The analysis of the source code is also often crucial for preservation, making it possible to re-create a work, or migrate it to different technologies. An experienced programmer familiar with a specific language will be able to find useful information about the functions being performed.

There are a few different techniques that can possibly be applied if original source code is available:

- The source code can be translated to Pseudocode, "an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading. (...)The purpose of using Pseudocode is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of an algorithm."[7]

- If the source code is available it may be possible to recompile or transpile it. In both techniques the original code is translated either to machine code (compiling) or to a different language (transpiling). There weren't any art-related examples of these techniques actively being used, and with the currently available tools and techniques a lot of resources would be necessary to use these techniques successfully.

- If no source code is available, in some cases it may be possible to decompile an executable file. The results are often unreliable, or may require extensive analysis to be understood, but depending on the technologies it may be worth an attempt. "Simple compiled script languages, such as Flash/SWF/ActionScript binary objects can be decompiled quite well. Other useful decompile scenarios (technically better termed 'disassembly') would be the preparation of 'binary patches', e.g. to understand and exchange specific external references which are hard coded and maybe don't exist anymore, breaking DRM (Digital Rights Management), or simply to understand specific routines" (Klaus Rechert).

[6] Deena Engel and Glenn Wharton, "Source Code Analysis as Technical Art History," *Journal of the American Institute of Conservation*, 54 – 2 (2015), 91-101.
[7] https://en.wikipedia.org/wiki/Pseudocode

Even if collecting and keeping source code accessible can be time-consuming, for many works it is nevertheless a sound step, as it opens a series of different options for preservation that would not be available otherwise. Institutions and conservators must assess their collections and artworks and assess if the source code is available, usable and relevant, and if it makes sense to collect it for their purposes.

When thinking of the preservation of a work, it is important to apply other techniques in parallel; we discuss this further in the following chapter.

Preservation Techniques: Disk Imaging, Emulation and Virtualisation

In this section the focus is on software-based artworks that currently run on specific hardware, as is the case with many interactive installations displayed in a Museum context. The strategies described here are already being systematically applied to different types of software-based artworks, more specifically in web-based or CD-ROM collections.

We agreed that creating a disk image of any computer hard-drive supplied with an artwork is an essential first step to safeguard the information contained in such a fragile object. This disk image can be stored as documentation of the running system delivered by an artist, and it can also be run on an emulator or virtualiser to partially test for technical dependencies and significant properties. A complementary technique is the creation of a new disk image to run an artwork's software. In this process, a new disk image is created within an emulation or virtualisation platform[8], the appropriate generic operating system is installed, and the artwork's software is then tested on that system. The most likely initial result is that some libraries or drivers will be missing, and this will be made very visible by error messages. Those missing libraries and drivers can then (depending on the age of the system and obscurity of the software) be added, or in some cases replaced, so that the artwork's software runs.

[8] QEMU would be an example of a platform that can be used both for emulation and virtualisation, while VirtualBox is a common Virtualisation platform.

The following step is to confirm that any significant or work-defining properties identified for a specific artwork can be maintained in the emulated version[9]. The best way to do this is to compare a native and an emulated version, but this will eventually become impossible once the original hardware is no longer available. This type of comparison is often a challenge and requires an understanding of the artwork at both a technical and conceptual level. Not only must the software be tested for its in- and outputs, which can mean testing sound, interactivity and image quality, but often the outputs are not audio visual at all. The main challenge is often to find, and recognise, the properties that are not obvious, and that you did not expect.

Nonetheless, if a disk image can be run correctly on either an emulation or virtualisation platform, and if it is comparable with an approved version of an artwork, then hardware dependencies can be considered to have been removed, or at least decreased, and the risks to the longevity of the software can be better managed. If any dependencies are identified during the process of emulation, then a future move to a different emulator or virtualiser is less likely to be problematic. [10]

The choice between emulation and/or virtualisation platforms is dependent on the technical requirements of an artwork and the present development of the emulation or virtualisation platforms. This is the primary reason why the choice for either virtualisation or emulation is often defined by the historicity of the material, and the type of CPU. The aim, which as previously outlined may not always be achievable due to age and/or obscurity of software and hardware, is to create disk images of an artwork's software systems that can be transferred between emulation and virtualisation platforms without loss. This would enable the identification of the essential technical dependencies of the original environment, and some of the long-term risks for preservation.

Virtualisation is driven by the information technology industry and its aim is to facilitate the management of current work environments and servers. "Virtualisers (such as VirtualBox and VMWare), are only able to run Intel-based x86 VMs" (Rechert 2016). Emulation, on the other hand is aimed at running older operating systems on newer platforms, and emulators cover almost any technical platform, in particular obsolete ones. Emulation has been gathering

[9] For more information about the 'work-defining properties' see Pip Laurenson, "Authenticity, change and loss in the conservation of time-based media installations," *Tate Papers* Issue 6, 2006, http://www.tate.org.uk/research/publications/tate-papers/06/authenticity-change-and-loss-conservation-of-time-based-media-installations

[10] For an in-depth analysis of emulation, including the differences between emulation and virtualization, and their limitations in terms of access to peripherals, see "Introduction to an Emulation-based Preservation Strategy for Software-based Artworks" (Rechert et al. 2016), a position paper on the topic based on the CoP discussions and the goals of PERICLES.

momentum as a preservation method, and new frameworks have been developed to support its use, by projects like the BwFLA[11], EMIL[12], the Olive Executable Archive[13] and the Internet Archive[14].

In general, emulation and virtualisation can be considered as medium-term solutions to preserve technical elements of many types of artwork, but the conceptual elements of an artwork must also be described and captured in other ways, which leads to the following discussion about documentation.

Documentation

There are many different ways how and reasons why to document a software-based artwork, and this section suggests some strategies that were found relevant and useful by the participants. Some of the key types of documentation fall under the following categories:

- **Documentation of the constituents of an artwork**
  Documenting the constituents of an artwork, both software and hardware, means having a basic knowledge of what the components of a work consist of, and their location and movements. This usually does not require expert knowledge but is essential in an institution.
- **Documentation of the production and display technologies**
  This type of documentation should identify the technologies of production and display of the artwork and identify risks for preservation, for instance dependency on specific hardware. It requires a deeper understanding of the technologies involved and often demands research into the history of a work and evolution of specific technologies.
- **Documentation of the installation process**
  This is particularly important for complex works that may require unique hardware, or calibration of software. A detailed installation manual, diagrams of technical connections, screen recordings of the work being setup or a video of the installation process may be essential to ensure that an artwork is displayed properly.

[11] http://bw-fla.uni-freiburg.de
[12] http://www.multimedia-emulation.de/index.html
[13] https://olivearchive.org
[14] https://archive.org/details/softwarelibrary_msdos_games

- **Documentation as reference**
  Documentation can be created in order to provide a reference points regarding how a work is supposed to look and function, or how it evolved through different iterations. This is particularly important for software-based artworks, because having software running does not mean that it is running correctly, nor is having the software running correctly always enough to ensure that an artwork is correctly presented. This type of documentation usually requires the use of video or screencasts, but virtualisation or emulation can also be used as a form of reference documentation.
- **Documentation of the visitor experience**
  It is important to show or describe how people experience the artwork, so that in the future the artwork can be contextualised. This can be done with video recordings of visitors interacting with a work, but also through interviews with those same visitors.[15]

All of the documentation listed above should, if possible, be defined in collaboration with the artist, through interviews, or more frequently through communication during the acquisition process, or ahead of a display of a work.

Without previous knowledge of the artwork and without additional information, it can be very hard to recognise if something is missing or just not working properly. Ideally the artist is involved in these steps, to ensure that details are correctly captured. If there is access to an approved version of the work on display, then a video documentation of the artwork running can be a good option for documentation.

For the purpose of conservation several guidelines have been created in the past to aid the creation of documentation (see appendix for some examples), these are not necessarily specific for software-based artwork, but relevant nevertheless. In general, the available documentation frameworks or guidelines can be useful as an aid to prompt relevant questions or highlight specific information. However due to the variability of software-based artworks, these generic resources should not be used as a standardised questionnaire but adapted to respond to the work being considered. Just as the artwork evolves, so will the thinking of the conservator and artist change over time about what is necessary to change and what must not be altered. Trying to adhere to a strict documentation framework presents the following limitations:

[15] The Daniel Langlois Foundation and the DOCAM project published a series of case studies on this type of documentation. Most relevant is Lizzie Muller, “Towards an Oral History of New Media Art,” Daniel Langlois Foundation for Art, Science, and Technology, 2008, http://www.fondation-langlois.org/html/e/page.php?NumPage=2096. Another interesting example is Rolf Wolfensberger, “Paul Sermon, *Telematic Vision* (1993 -). Documentary Collection,” Daniel Langlois Foundation for Art, Science, and Technology, 2009, http://www.fondation-langlois.org/html/e/page.php?NumPage=2179.

- A rigid template makes it difficult to fit something in the scheme that has not been previously encountered;
- The type of information to be included is likely to be either very general or extremely detailed;
- It can be difficult to represent relations between different elements;
- It does not reflect the iterative nature of documentation practice as a work evolves over time, and in relation to different activities it will encounter such as being displayed in different spaces and contexts, conservation activity and also research.

Partly due to these challenges, most conservators have come to rely on their own experience, rather than actively using standardised documentation frameworks that may require information irrelevant for a specific artwork while omitting information essential to a software-based artwork. Conservators, or the teams responsible for these works, strive to have the information necessary to manage works in a collection, for the re-installation of each artwork, and their preservation for the foreseeable future. This is a significant challenge, particularly when considering the rapid development and evolution of technology. A useful documentation framework should clearly define the objective of the documentation, reflect the iterative nature of documentation, and accommodate change.

Video as a documentation tool

Video has been used to document complex artworks with a temporal aspect for as long as video exists, albeit initially this meant recording human performances, rather than software-based artworks. Software-based artworks can display a wide variety of behaviours, the only limitation to those being the in- and output devices and interfaces. These behaviours are determined by the artwork's software, including programming errors, bugs and technical limitations encountered at time of production. The whole underlying system of software and hardware is often of integral importance, to run the software but also to understand its production history. Settings and interfaces will also influence how a software-based artwork manifests itself. Given all these variables it can be difficult to understand whether a software is behaving as it was meant to, or designed to do. This is a characteristic particularly relevant to software-based art, and documentation must be adapted to reflect this. Traditional conservation documentation, listing components and technical specifications, is often inadequate, and even thorough display instructions can be insufficient.

The key point discussed was that video is essential to document not only the behaviour of these works, but also their installation processes. This can mean recording different hardware components being connected or screencasts demonstrating how to set-up an artwork's software. For example, if a work is being installed for the first time in an institution, video documentation of a previous installation process and a video of the work running are likely to be more helpful than a documentation folder. The caveat here is that even a screencast of the software set-up may be insufficient, as it will not necessarily show the whole software environment, namely underlying dependencies, like for instance the version of DirectX. It must be recognised that there is a limit to what video, as a linear medium, can capture. For instance, video is unlikely to fully capture the behaviours of very complex, interactive, or variable works.

Implicit Knowledge

With variable and complex artworks, it can be hard to judge what type of information has to be captured, or the level of detail one needs to document. These two factors also depend on the purpose of the documentation being created. It is always easy to miss implicit information, whether technical or non-technical. It is the obvious and trivial that is often neglected.

For example, the way a mouse is used, or what keys on a keyboard are pressed, are very obvious behaviours for current users of computers, but in the future, once mice are no longer used as a standard, it will be very important to understand their use, and artworks may have to be adapted to accommodate this change in user habits. Another instance is the use of a scrolling bar in the work *Scrollbar Composition* by Jan Robert Leegte[16], which did not pose an obvious problem until scrolling bars stopped being common features on web browsers.

Also, sometimes there are implicit external dependencies in a software environment that are not necessarily described, such as the Internet Protocol (IP), which have evolved over time. Although some software may only work in a specific environment, when such context is not documented or directly visible, it is easily forgotten. Next to hardware dependencies, data protocols can also be important to document; for example, network latencies or how a network is set up can affect how works run. Because the stability and speed of the connection to the network is often variable, it is thus crucial to understand the protocols and the definitions of how the software communicates with any web resources. In other words, documenting software-based art is ideally also about documenting context: the context of creation, as well as

[16] http://www.scrollbarcomposition.com

the behaviour, the environment in which the works functions, and the social and technical historical context.

Capturing Change

Following initial work carried out by Deena Engel and Mark Hellar, an approach for documenting change that is starting to be discussed and tested in museums is the use of Git.[17] Git is a version control system, commonly used in software development. Using this type of version control tools allows artists and collecting institutions to keep track of changes to an artwork's software throughout its life. It also allows for a return to earlier versions of the software, if a change has unexpected consequences or is unwanted.

Museums have more experience of preserving things that change much more slowly due to material deterioration, such as charcoal drawings or marble sculpture. With software-based artworks there are often deep changes at different moments of the artwork's life, and such changes can be made by the artist/programmer and/or the museum. These changes can be very difficult to capture, as they often take place under tight deadlines (typically before an exhibition opening) and are typically carried out by the artist or a programmer outside the institution. By using Git, code changes are documented, and can be undone, according to the artist and the museum's need.

Further experience and research from conservation practitioners using tools such as Git for tracking and documenting change within software based artworks will help to evaluate how they might be used to create valuable documentation relevant to preservation. In the end though, Git is only a tool for documentation, so its content and the value for a collecting institution will depend on its correct use.

[17] Deena Engel and Mark Hellar, "Computational Provenance and Computational Reproducibility: What Can We Learn About the Conservation of Software Art From Current Research in the Sciences?" (paper presented at EMG (Electronic Media Group of the American Institute of Conservation of Historic and Artistic Works), Miami, Florida, May 16, 2015).

Case Study: Dependency on external data

One specific risk for preservation that was identified very early on in the discussions, was related to works that depend on live feeds from specific sites, such as chat rooms[18], Twitter feeds, Google searches[19], or Google Maps. These types of data sources raise a series of both technical and ethical questions and we will briefly discuss those here. It is relevant that these data sources are beyond the control of the artist or the museum, and that they usually display live results. In most cases the connection to the online data happens through an Application Programming Interface (API), so not only must the source website be live, but it must also still be using that specific API.

From the discussions it became apparent that many artworks that use live feeds often rely on a database to store the collected data. This intermediate database is then queried by the artwork's software. This is usually a practical solution to issues related to slow/lack of Internet connections and the overall reliability of a work when on display. This process means an artwork can be run even without live access to data, and also that the data that is captured is stored. Adding a database can have a series of advantages for conservation, so even if a database is not initially part of the system, one can be added to record live data sets.

Preserving such data sets means a historical version of the artwork is captured through the saved results, and allows their use for exhibition in the future, should the API no longer work. The implication is that the 'live', contemporary aspect of an artwork may then be lost. For example, a work that is meant to "react" to current events might lose meaning if using preserved data. However, some artists might feel that data sets should remain static in the future as the work (and thus the crawls) represents a certain era of the Web.

If the 'liveness' of the results is essential, then the other possible approach is to change the information source. For instance, instead of gathering communication from Google, one may decide to switch to a different search engine. If a similar platform exists this change can be seen as straightforward, but in some cases this may mean an important compromise.

[18] For instance *Listening Post* (2001), by Mark Hansen and Ben Rubin, relied on IRC chat rooms, for a discussion of the piece's evolution over the last 14 years, see: http://modes.io/listening-post-ten-years-on.
[19] An example from the Tate Collection is *Brutalism: Stereo Reality Environment 3*, 2007 (T13251) by José Carlos Martinat Mendoza.

A tool that is being developed to capture online data with the objective of documenting results is the Webrecorder. Developed by Ilya Kreymer and Dragan Espenschied at Rhizome[20], this tool is currently meant to be used for documenting websites, but can also record web services like Google Maps, which might provide essential elements to an artwork.

Case Study: Reconsidering previous preservation efforts

Paul Jansen Klomp was invited by the CoP to present a case study on the preservation of the work *Revolution. A Monument for the Television Revolution* (1990) by Jeffrey Shaw and Tjebbe van Tijen and how the approach initially used in 2006 became obsolete and was updated in 2015. *Revolution* was originally produced for the travelling exhibition *IMAGO: fin de siëcle in Dutch contemporary art,* a co-production between the Netherlands Office for Fine Arts (now RCE) and the Netherlands Media Art Institute (now LIMA) and is now in the collection of the Cultural Heritage Agency of the Netherlands. Within the scope of the European project *Inside Installation* (2004–2007) Paul Jansen Klomp, in collaboration with the Netherlands Media Art Institute (now LIMA), used Pure Data to re-create the work.[21]

The installation by Shaw and Van Tijen is fairly simple; visitors can push an extended steel bar that is attached to a steel column, which holds a built-in monitor. When the bar is rotated, and the machine moves, the images on the monitor change accordingly. “Pushing the bar forward triggers 180 images depicting revolutionary moments in human history. Rapidly turning the bar produces a vague blur of images, and pulling the bar backwards results in an image of two millstones grinding corn” (Wijers, 85). As Klomp recalls, at the time in 2007, the work seemed easy to document and emulate with merely one single channel and repetitive sounds that only changed to the speed of the turns.

To begin the process, Klomp noted down data about the working of the piece, as the original was still available and in working condition. He measured inputs and outputs, and the response times. In the meantime the original computer failed due to battery leakage. Replacing the PC proved difficult, no similar PC would work. Only after finding that the original computer had been hardwired by the artist or his technician was it possible to run the program again. This was done on the original computer, which was repaired by adding an I/O card to overtake the function of the damaged components on the motherboard.

[20] https://webrecorder.io
[21] Pure Data is an open source visual programming language for creating interactive computer music and multimedia works. For more information, see http://puredata.info

> “We didn’t get into the code of the original program. We considered the computer and all the other devices to be a black box and we just monitored the input and the output. Based on this monitoring we made the simulation.” (Paul Jansen Klomp)

The choice for Pure Data was made because it was open source and could be used in a Linux environment, thus not requiring a dependency on any licensing from external parties. Another reason was that the installation worked with sound in different playback speeds and it was very easy to use in real time with Pure Data. The re-created version was considered successful as the behaviour was very close to the original. However, in 2015 Klomp tried to run the recreated version, but it refused to work due to changes in Pure Data. Running the Pure Data version on a VMWare virtual machine also proved impossible, as Pure Data would not connect correctly to the audio output. A few different strategies were tested; running the old Ubuntu version in a virtual machine; running the original Pure Data version in a virtualized Windows XP; migrating to a newer version of Pure Data in Windows 10; all attempts were unsuccessful. After all these experiments, Klomp solved the sound problem as it turned out that it was related to a setting in the host computer: whether or not the sound drive would play multiple sound streams. When turning off the multiple sound streams option, Pure Data in the virtual machine (VMWare) could connect to the computer again and the sound worked as before.

While the sound problem was resolved, one issue remained unclear: how does the VMWare player handle USB connections? When there is a USB connection from a host system, where should the driver be: should a driver be re-installed in VMWare or can the driver be in a host computer? Perhaps the answer is documented in the VMWare environment, but so far Klomp has been unable to find it. In the long term, the dependency on the RS232 communication over USB to communicate to the sensor (which is in a separate box with its own micro-controller), could become a problem, so to overcome this Klomp migrated the work into Windows 10. This meant that the Pure Data programme could stay intact and only the video needed to be re-encoded to a current Windows media standard.

Whereas in 2007 they expected that the emulated version inside the virtual machine would easily start after a few years, less than ten years later it turned out that there was neither a working emulation nor a working copy. In conclusion, Klomp remarked that in hindsight he would not choose to work in Pure Data anymore, but at the time one’s choices are driven by what is available. The challenge of working with open source initiatives is that one is very dependent on the energy that a community puts into sustaining and maintaining changes. For a future project he would rather use C++ open frameworks for example and program it in a more

traditional fashion, so that it potentially would be more stable in the long-term. The experience also showed the importance of carefully checking the dependencies of the audio hardware, in similar ways to the other dependencies.

Klomp's challenges were recognisable, for example, the difficulty of tracking changes or revisions in Pure Data, as well as the dependency on a serial connection. In general, in reports and manuals on emulation only the use of software that runs on standard systems is supported, but as the case of *Revolutions* showed this becomes difficult to interpret since specific custom-made changes were made that cannot easily be replicated within the standard emulation technology.

To conclude, for the time being, it is possible to emulate software, but hardware dependencies may require custom solutions or be impossible to emulate. Emulators are made to work with a limited number of standard components that replace the hardware components. If an artwork requires a non-standard component then other solutions must be found. For example, once physical connectors (like RS232) and the protocols they use are out of date, many layers come in between the hardware and the code, which complicates the use of standard emulation technology. Only if the software part can be isolated from the interfaces, can it be emulated without a problem. To facilitate a decision-making process, it would be valuable to identify any external dependency as soon as possible.

Concluding remarks

Throughout our discussions, a recurring theme in our findings involved the difficulties associated with capturing sufficient and relevant information about software-based artworks. This information should enable the display and preservation of these works. It became clear that merely keeping the source code of an artwork is not sufficient to enable the recreation of a work at a later stage; understanding the technical specificity of each work and capturing its performance is just as important. This is not to say that a close reading of source code is not helpful or important, but careful planning is needed to keep specific connections intact over time, especially when a work relies on multiple dependencies between hard- and software. While emulation and virtualisation are valuable methods to research a work's behaviour and functioning, these methods pose their own challenges in relation to obsolescence, and more effort is needed to adapt these tools for the preservation field. It is important to create a better understanding of cultural and technical frameworks to aid decision-making processes in order to

be able to interpret and understand software-based artworks in the future.

The challenges that came up in the discussions underscored the need for a professional network and highlighted the value of being in contact with one another to those involved; sharing experiences and knowledge to those involved. This exchange opens the possibility to experiment with different approaches and reflect on each other's practices. These experiments could become a way to work towards a shared practice across different organisations, leading to the development of new conservations strategies. What became very clear is that whilst understanding and preserving software-based artworks must happen within the context of each individual organization, it has been tremendously helpful to have these broader discussions. This is especially so given the new territories of knowledge and the challenges that software-based artworks pose, as well as the overall scarcity of resources and protocols. The discussions in this community of practice group have contributed to the development of both conservation practice for software-based artworks and, and perhaps most crucially, an international community who are actively engaged in their conservation.

Appendix

Some suggestions for further reading

On computation

Paul Ford. "What is Code?" Online: http://www.bloomberg.com/graphics/2015-paul-ford-what-is-code.

Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software*. Redmond, WA: Microsoft Press, 2000.

On emulation and virtualisation

Klaus Rechert, Patricia Falcao and Tom Ensom. "Introduction to an Emulation-based Preservation Strategy for Software-based Artworks." 2016.

Patricia Falcao, Annet Dekker, Pip Laurenson, "An Exploration of Significance and Dependency in the Conservation of Software-based Artworks" (paper presented at EMG (Electronic Media Group of the American Institute of Conservation of Historic and Artistic Works), Miami, Florida, May 16, 2015).

Gaby Wijers, "To Emulate or Not? Conservation Case Studies From the Netherlands," in *Inside Installations. Theory and Practice in the Care of Complex Artworks*, eds. Tatja Scholte and Glenn Wharton (Amsterdam: Amsterdam University Press, 2011), 81–90.

On source code

Deena Engel and Mark Hellar, "Computational Provenance and Computational Reproducibility: What Can We Learn About the Conservation of Software Art From Current Research in the Sciences?" (paper presented at EMG (Electronic Media Group of the American Institute of Conservation of Historic and Artistic Works), Miami, Florida, May 16, 2015).

Deena Engel and Glenn Wharton, "Source Code Analysis as Technical Art History," *Journal of the American Institute of Conservation*, 54 – 2 (2015), 91-101.

Deena Engel and Glenn Wharton, "Reading between the lines: Source code documentation as a conservation strategy for software-based art," *Studies in Conservation*, 59 – 6 (2014), 404-415.

On conservation and documentation of software-based art

Electronic Arts Intermix, resource guide:
http://www.eai.org/resourceguide

Digitising Contemporary Art. 2013. Guidelines for a Long-term Preservation Strategy for Digital Reproductions and Metadata.
http://www.dca-project.eu/images/uploads/varia/DCA_D61_Guidelines_Long_Term_Preservation_Strategy_20120213_V1.pdf

The Daniel Langlois Foundation for Art, Science, and Technology
http://www.fondation-langlois.org/html/e

DOCAM - Documentation and Conservation of the Media Arts Heritage
http://www.docam.ca

POCOS - Preservation of Complex Objects Symposia
http://www.pocos.org

Tech Focus III:
http://resources.conservation-us.org/techfocus/techfocus-iii-caring-for-computer-based-art-software-tw
Transformation Digital Art, preservation of born-digital art (September 2016)
http://www.li-ma.nl
Webrecorder, Rhizome:
https://webrecorder.io

Examples of documentation models

Matters in Media Art:
http://mattersinmediaart.org
PREMIS:
http://www.loc.gov/standards/premis
Variable Media Questionnaire:
http://variablemediaquestionnaire.net

Platforms using emulation

bwFLA- Emulation as a Service:
http://bw-fla.uni-freiburg.de
Olive Executable Archive:
https://olivearchive.org
Internet Archive:
https://archive.org/details/softwarelibrary_msdos_games

This project has received funding from the European Union's Seventh Framework Programme
for research, technological development and demonstration under grant agreement no 601138.