Conserving Digital Art for Deep Time

Francis T. Marchese

Francis T. Marchese
Educator, Gallery Director
Pace University
Computer Science Department
New York, New York 10038
USA
fmarchese@pace.edu

ABSTRACT

Displaying digital art in the late twentieth and early twenty-first centuries is already proving to be a challenge. Exhibiting this same art in the distant future will depend upon new thinking and practices developed today by artists, conservators, and curators. Established software engineering methods for dealing with aging systems can provide a new model for the conservation of digital art, and a foundation for the enhancement of art-historical scholarship. Artists with an interest in a more refined approach to the programming that underpins their work will also be interested in software engineering concepts.

Introduction

Consider the following scenario. It is the year 2511, and a museum curator of 21st century digital art plans an exhibition encompassing the first decade of the 21st century. This decade was a heady time for digital art. Relentless innovations in computer hardware and software, and nearly limitless access to information afforded by the World Wide Web, provided artists with the means to create singular and collaborative artworks. Works ranged from the purely ephemeral, such as performance, to those fixed to a specific physical substrate, such as an immersive installation. Independent web sites, such as Rhizome.org and Turbulence.org, commissioned and hosted digital art, complemented by online museum galleries, such as the Whitney Museum's AirPort. New York's Museum of Modern Art (MoMA) and the Tate Modern in London increased their rate of acquisition of digital art through the 21st century's first decade, although these entities were far outpaced by Ars Electronica, which by 2010 had amassed over 30,000 digital artworks [1].

By 2010 it had already been recognized that digital art would pose many challenges for museums, particularly in terms of maintaining its longevity. General approaches to digital preservation had been focused primarily at the institutional level, which considered an organization's goals, priorities, available resources, and management policies [2]. A digital object within this context became a discrete entity with well defined attributes that could be managed throughout its lifetime. The goal of the digital preservation community had been to create standards and develop best practices for the conversion of digital material into "archival" formats that could be manipulated and shared.

Digital art conservators had taken a two-pronged approach to preserving digital art as explicit digital artifacts: technology preservation and document compilation. Technology preservation encompassed both physical and digital artifacts. In the former case, computer technology is stockpiled to support the artwork in the inevitable case that a component fails (e.g., CPU, hard drive, etc.). In the latter, digital artifacts such as computer programs or digital videos must be archived to durable media. Because any digital storage medium (e.g., tape, CD, DVD) ultimately either decays or becomes obsolete, all digital artifacts must be routinely refreshed to a new storage medium. In document compilation, an extended set of documentation is assembled to help define and contextualize the artwork with the express purpose of making the artwork

displayable at some future date. Artist interviews, questionnaires [3], artist-conservator-curator collaborative discussions, conservation workshops [4], and outright documentation of a program's source code were all approaches that had been taken. Yet these procedures remained experimental in nature, and had not been integrated into the formal scientific activity of art conservation [5].

Meanwhile, computer technology continued to advance, with older technology fading into obsolescence. Concomitantly, the variety of computer languages and software employed by digital artists expanded. In 2010 alone, languages such as C/C++, Java, JavaScript, Flash, HTML, XML, Processing, Perl, and Max/MSP were in use with widespread exploitation of development libraries and environments for such things as mobile application development, sound composition, virtual worlds, and computer games. Operating systems evolved, database formats changed, and globally accessible data disappeared or became redistributed. Hence, given the assured evolution or transience in even the simplest of digital entities (e.g., video file formats), it would be difficult to predict that a computer-based artwork created in 2010 would survive intact so it could be exhibited, as originally constructed, in the year 2060 – certainly not in the year 2511!

Digital Art Conservation

Artworks collected by museums possess a particular magisterial quality. Because museums are the *de facto* keepers of cultural heritage, any work acquired by a museum is expected to become part of the canon. Preservation is an essential part of museum practice. Once an institution has decided to acquire an artwork, its evaluation and care is entrusted to the museum's conservators.

The practice of art conservation is a formal scientific activity, defined in the following way by the International Council of Museums Committee for Conservation:

> The activity of the conservator-restorer (conservation) consists of technical examination, preservation, and conservation-restoration of cultural property: Examination is the preliminary procedure taken to determine the documentary significance of an artifact; original structure and materials; the extent of its deterioration, alteration, and loss; and the documentation of these findings. Preservation is action taken to retard or prevent deterioration of or damage to cultural properties by control of their environment and/or treatment of their structure in order to maintain them as nearly as possible in an unchanging state. Restoration is action taken to make a deteriorated or damaged artifact understandable, with minimal sacrifice of aesthetic and historic integrity. [6]

Traditional conservation practice thus focuses on an artwork as an integrated physical whole, the integrity of which must be preserved. Change is defined as a process that will deleteriously affect the stability of an artwork, moving it away from its original reference state and altering its identity.

Time-based digital artwork does not fit this definition because change is an intrinsic part of its nature. Museum conservators in charge of maintaining time-based media, that is, artwork whose aesthetic experience evolves over time, realize this and are attempting to expand the conservation paradigm to accommodate digital art. Pip Laurenson, Head of Time-based Media Conservation at the Tate Museum, has proposed a redefinition of conservation practice to accommodate time-based media, so that conservation becomes the means by which an artwork's essential properties are documented, understood, and maintained. Its aim is the preservation of the artwork's *identity*, so that it may be displayed in the future as different possible authentic installations [7]. For Laurenson, the *identity* of a digital work should be considered as a

collection of properties which include: the artist's instructions, approved installations intended to act as models, an understanding of the context in which the art was made, and the degree to which the artist specifications reflect his or her practice at the time the art was created. For a 26th century conservator this means that if the standard methods of digital art conservation (e.g., migration and emulation) eventually fail, then the preservation strategy of reinterpretation, that is, the process of recreating part or all of the artwork utilizing this extended documentation, can be invoked.

This expanded notion of an artwork's identity as a collection of concepts and artifacts may be used as a starting point for formal identification and documentation of an artwork employing the principles and practices from the field of software engineering. Software engineering is the process of applying a systematic, disciplined, quantifiable approach to problem analysis, system and software design, its development, operation, and maintenance [8]. Software engineering methodologies focus on both the software product and the process used to create and maintain it. In the latter case, the software life-cycle is an extension of the business life-cycle and is defined by the business process management model (BPM) [9]. As such, its tools and techniques may be integrated into a museum's conservation practice. In the former, software engineering artifacts may be adapted to the representation of a digital artwork. Software engineering as a process may engage all stakeholders who comprise an art museum's business practice, including artists, curators, conservators, installers, maintainers, museum directors, art historians, and viewers; and can reflect and integrate this process into a museum's current best practices. Finally, software engineering practices may be adapted to all digital conservation strategies, including refreshing, migration, emulation, and recreation.

Documentation and Conservation

When a museum acquires a digital artwork, it is usually assumed that the work will be in a form that will allow it to be exhibited as is, with no further developments, enhancements, or adaptations expected from the artist who created the work. This work, along with its underlying concepts and technologies, is now frozen in time. In computing this is known as a legacy system [10]. Assessing the state of a legacy artwork at some point in time may be difficult for any number of reasons. For example, the artwork's existing documentation may be incomplete. Or changing museum business processes (e.g., curatorial milieu) which supported the original acquisition and installation of the artwork may be directed in the future along other trajectories, thus making upkeep of some legacy artworks more difficult. The nature of the digital artwork itself may be difficult to assess. It may have been assembled from many diverse components without a consistent design or programming style. Or it may be in either executable form or written in an obsolete programming language.

Strategically, there are four approaches for dealing with traditional legacy systems. The first is to scrap the system outright, as business practices have changed and the system is no longer needed. If the system continues to work well and can be easily adapted over time, continued maintenance of the software is possible. But if system performance or usability continues to degrade over time, then the system or its parts must be transformed to improve maintainability. Finally, a system must be replaced if obsolete hardware or software precludes further operation, or the new system can be built at reasonable cost.

For conservators, each one of these approaches creates its own set of issues, with degrees of intervention from minimal adaptation to full restoration, many of which are related to the importance of the artwork – an importance that should be expected to change over time. This is why it is most important that the long-term process of conserving a digital artwork

must be based on a thorough understanding of its structure or architecture, as represented by a breadth of documentation.

Documentation is an intrinsic component of any system. Software engineering provides a systematic methodology for creating and maintaining documentation that supports communication, preservation of system and institutional memory, and processes such as system auditing. Within this context a computer system's documentation should provide comprehensive information about its capabilities, architecture, design details, features, and limitations. It should encompass the following components:

Requirements – Statements that identify the capabilities and characteristics of a digital artwork. This is the conceptual foundation for what has been created.

Architecture/Design – An overview of software that includes the software's relationship to its environment and construction principles used in design of the software components. Typically a system's architecture is documented as a collection of diagrams or charts that show its parts and their interconnections.

Technical – Source code, algorithms, and interfaces are documented. Comments may be embedded within the system's source code and/or parts of external documentation.

End User – Manuals are created (e.g., static documents, hypermedia, training videos, etc.) for the end-user, system administrators, and support staff.

Supplementary Materials – Anything else related to the system. This includes: legal documents, design histories, interviews, scholarly books, installation plans, drawings, models, documentary videos, web sites, etc. [11]

Each component is important to the representation of a digital system. Each may operate at a different level of abstraction or within a particular context. *Requirements* documentation presents the conceptual view of *what* the system is expected to do. It is written to be understood by all the stakeholders who comprise the art museum's business practice. *Architecture/Design* documentation functions very much like an architect's sketch of a building, showing all its components and how they fit together. *Technical* documentation represents the bricks-and-mortar of the artwork, conveying information about how the artwork is constructed.

Besides facilitating an artwork's conservation, this documentation could also support scholarship, enabling art historians to understand an artist's working process and evolution of practice. A computer system's structure is a reflection of the conceptual space the artist had been working in at the time the art was created. The number of software components, their hierarchy, and the interconnections among them should give an idea of how the artist viewed a representation problem, and how it was transformed into a computer system. An analysis of documentation should yield answers to questions about:

Authorship – Who wrote what? (Was it by the hand of the artist, or was it built by others?)

Educational/Cultural Context – Who influenced the artist conceptually or technologically?

Craftsmanship – How well was the program written and system built?

Aesthetics – How well conceived and designed is the software? Does it possess an elegance and refinement comparable to any other beautifully created object?

Development Process – What were the design strategies used by the artist?

Technical Context – What were the development tools available at the time the artwork was created?

Theoretical Foundation – What theories of computing did the artist use?

Documentation and Digital Artists

Documentation is a fundamental part of a digital artwork. A program's source code and the data it uses are *de facto* documentation, along with directions for its installation. The question remains – how much more documentation is required to provide a sufficient representation of a digital artwork? This is an open question for any software engineering project, as it is for a digital artwork, and depends on factors such as project size, complexity, and expected system lifespan. It may be argued that it is not the artist's responsibility to provide sufficient documentation for an artwork, but if contemporary curatorial practice is an indication of what the distant future will hold, the following scenario is most likely for a year 2511 exhibition. Artwork selection will not only be based on its importance to the canon, but also the availability of resources (e.g., staffing, time, funds) required for its installation. An artwork that may be the best example of a theme or idiom may need to be replaced in an exhibition by a lesser work, because its own documentation proves insufficient for its recreation.

It is posited here that a digital artist shares a certain responsibility for the long-term preservation of an artwork. Traditional artists who employ archival media and follow well established best practices for creating a stable physical artwork produce works with a high probability of standing the test of time. For those artists who consciously choose to work in a non-archival way, the long-term preservation and ultimate exhibition of their works is a problem with an indeterminate solution. It should be remembered that major museums possess culturally significant artworks that cannot be exhibited because of their fragility or degree of deterioration.

Digital artists have choices analogous to their traditional counterparts for maintaining their artwork's longevity. Best practices exist for the development of computer software in different ways. For example, the use of standard programming styles, data structures, algorithms, and the selective insertion of comments into source code represent programming best practices. Software engineering best practices work at a different level. The software engineering conceptual process of analyze-design-build is consistent with artistic practice. And the tools used by software engineers during the analysis and design phases of software development allow software designers literally to sketch out a system's architectural design. Including these design representations with an artwork's source code expands its representational details to encompass the *Requirements*, *Analysis/Design*, and *Technical* documentation categories discussed above. It should provide as well a sufficient description of the artwork's *identity* to allow it to be recreated at some future time if all other conservation strategies fail.

Back to the Future

One of the works selected for the 2011 retrospective by the 26th century curator was *Trigger*, a site-specific, sensor-activated, immersively projected, interactive art installation by Jody Zellen that debuted at the Pace Digital Gallery in the fall of 2005 [12]. Zellen created *Trigger* to explore the transient stories that emerge from our relationships with urban spaces. The gallery was filled

with overlapping videos from seven projectors and infused with transient sounds. When visitors passed through motion sensors, sounds and videos changed, evoking the ephemeral nature of urban space, and the fleeting and shifting perceptions of it.

Neither *Trigger*'s original hardware nor software made it to the year 2011, much less to 2511; the system was disassembled and repurposed at show's end. The documentation available to the 26th century curator included: the artwork's original videos, a video walkthrough of the artwork, images of the installation, a catalog, and a short technical paper about the artwork [13]. The technical paper provided key information for *Trigger*'s recreation – its functional requirements, architecture, and design. The artwork's functional requirements identified its capabilities and characteristics, simply put: what the artwork was supposed to do. They were recorded in the technical paper as:

- The artwork would support a large number of projectors, sensors, and speakers.
- The artwork's sensors would capture viewer motion.
- Captured motions would trigger events.
- Events would be communicated to the artist's multimedia programs.
- The multimedia programs would change the content of the displays and alter sounds.

Trigger's architecture communicated the high-level interrelationships among its components without specifying the processing details. *Trigger* possessed a simple architecture of three loosely coupled components: a microcontroller-sensor system, application software, and interface software linking the sensor system to the application. Each of these components worked independently, communicating by simple message passing. Such a relationship exhibited the attribute of *low coupling*, a fundamental software design paradigm. Finally, the technical paper briefly described each component's responsibilities. In all, *Trigger*'s technical discussion covered no more than a page, but, combined with its supplemental materials (e.g., video walkthrough and installation images), that was sufficient for the 26th century recreation of the artwork to maintain its *identity* and reflect Zellen's artistic practice.

Trigger's clearly defined functional requirements and simple architecture made its recreation possible. It afforded the curator's team the flexibility of selecting the best computer method for reconstructing the artwork. In contrast, if *Trigger* had been conserved at the source code level, the curator's team would have had to sift through the code to extract its internal workings, and then either adapt the code to its 26th century environment, or reconstruct its architectural representation. In the end, recreation of *Trigger* became a trivial matter, utilizing a 26th century equivalent of a 21st century data-flow programming language for multimedia applications like Cycling 74's Max/MSP. Data-flow programming languages have had a long history, evolving from workstation-based scientific visualization systems such as AVS [14] in the late 1980s. Recognized for their ease and versatility in defining parallel event-based systems, these visual programming languages continued to evolve over time. Indeed, the original developers of *Trigger* had considered using Max/MSP in 2005, but were constrained from doing so.

Ultimately, the retrospective's curator capitalized on each artwork's unique documentation to craft a cogent story of early 21st century art. From *Trigger*'s high-level designs to another artwork's creatively commented source code, all of the artworks' documentation acted in

concert to help recreate a sense of what digital art was like at the dawn of the 21st century.

In Sum

The display of contemporary digital art at some distant future time remains an open problem, the solution to which will come from efforts by artists, conservators, and curators. It has been proposed here that the use of software engineering practices will provide a new jumping-off point for transitioning from the conservation practices used for traditional art to methods more appropriate for time-based media. This process will aid digital art scholarship as well, by organizing an artwork's components in such a way as to enhance accessibility by art historians. Finally, digital artists who choose to adapt software engineering practice to their artistic process will be able to extend the lifespan of their artwork.

References and Notes

1. C. Becker, et al., "Preserving Interactive Multimedia Art: A Case Study in Preservation Planning," *Lecture Notes in Computer Science*, Vol. 4822, 257–266 (2007).
2. NINCH Working Group on Best Practices, "The NINCH Guide to Good Practice in the Digital Representation and Management of Cultural Heritage Materials Online: The National Initiative for a Networked Cultural Heritage" (2002), retrieved January 10, 2011 from www.nyu.edu/its/humanities/ninchguide/.
3. J. Ippolito, "Accommodating the Unpredictable: The Variable Media Questionnaire," *Permanence Through Change: The Variable Media Approach* (New York: Guggenheim Museum Publications and Montréal: The Daniel Langlois Foundation for Art, Science, and Technology, 2003).
4. ERPANET, "The Archiving and Preservation of Born-Digital Art Workshop: Briefing Paper," The ERPANET Workshop on Preservation of Digital Art (2004). Retrieved January 10, 2011 from www.erpanet.org/events/2004/glasgowart/briefingpaper.pdf.
5. For a concise review of the current state of the problem, see T.A. Yeung, S. Carpendale, and S. Greenberg, "Preservation of Art in the Digital Realm," *The Proceedings of iPRES2008: The Fifth International Conference on Digital Preservation* (London: British Library, 2008).
6. International Council of Museums Committee for Conservation, "The Conservator-Restorer: A Definition of the Profession, Section 2.1." Retrieved January 10, 2011 from www.icom-cc.org/47/about-icom-cc/definition-of-profession/.
7. P. Laurenson, "Authenticity, Change and Loss in the Conservation of Time-Based Media Installations," *Tate Papers*, Autumn (2006). Retrieved March 15, 2011 from www.tate.org.uk/research/tateresearch/tatepapers/06autumn/laurenson.htm.
8. R.S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th Edition (New York: McGraw-Hill, 2005).
9. R.K. Ko, "A Computer Scientist's Introductory Guide to Business Process Management (BPM)," *Crossroads*, Vol. 15, No. 4, 11–18 (2009).
10. J. Ransom, I. Sommerville, and I. Warren, "A Method for Assessing Legacy Systems for Evolution," *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering* (CSMR 98), March 8–11, 1998 (Washington, DC: IEEE Computer Society, 1998) 128.
11. For a complete discussion, see C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd Edition (Upper Saddle River, NJ: Prentice Hall, 2005).
12. J. Zellen, *Trigger*, Pace Digital Gallery, New York, NY, October 18–November 8, 2005. Retrieved March 15, 2011 from www.jodyzellen.com/pace2.html.
13. F.T. Marchese, "The Making of Trigger and the Agile Engineering of Artist-Scientist Collaboration," *Proceedings of the Tenth International Conference on Information Visualization: IV'06*, London, July 2006 (Washington, DC: IEEE Press, 2006) 839–844.
14. C. Upson, et al., "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, Vol. 9, No. 4, 30–42 (1989).