An Archive of Interfaces: Exploring the Potential of Emulation for Software Research, Pedagogy, and Design

DANIEL CARDOSO-LLACH*, Computational Design Laboratory, Carnegie Mellon University, USA
ERIC KALTMAN*, Computer Science, California State University Channel Islands, USA
EMEK ERDOLU, Computational Design Laboratory, Carnegie Mellon University, USA
ZACHARY FURSTE, Carnegie Mellon University Libraries, USA

This paper explores the potential of distributed emulation networks to support research and pedagogy into historical and sociotechnical aspects of software. Emulation is a type of virtualization that re-creates the conditions for a piece of legacy software to operate on a modern system. The paper first offers a review of Computer-Supported Cooperative Work (CSCW), Human-Computer Interaction (HCI), and Science and Technology Studies (STS) literature engaging with software as historical and sociotechnical artifacts, and with emulation as a vehicle of scholarly inquiry. It then documents the novel use of software emulations as a pedagogical resource and research tool for legacy software systems analysis. This is accomplished through the integration of the Emulation as a Service Infrastructure (EaaSI) distributed emulation network into a university-level course focusing on computer-aided design (CAD). The paper offers a detailed case study of a pedagogical experience oriented to incorporate emulations into software research and learning. It shows how emulations allow for close, user-centered analyses of software systems that highlight both their historical evolution and core interaction concepts, and how they shape the work practices of their users.

CCS Concepts: • **Applied computing** → **Computer-aided design**; • **Human-centered computing** → **Empirical studies in interaction design**; *Interaction design theory, concepts and paradigms*; *Interface design prototyping*; • **Information systems** → **Digital libraries and archives**.

Additional Key Words and Phrases: emulation, pedagogy, architecture, computer-aided design, autocad, software history

ACM Reference Format:

Daniel Cardoso-Llach, Eric Kaltman, Emek Erdolu, and Zachary Furste. 2021. An Archive of Interfaces: Exploring the Potential of Emulation for Software Research, Pedagogy, and Design. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 294 (October 2021), 22 pages. <https://doi.org/10.1145/3476035>

1 INTRODUCTION

The historical study of software systems poses unique research challenges as these systems depend on networks of contemporaneous hardware and software that are technically difficult, and in some cases impossible, to reproduce. While oral and archival sources offer invaluable insight into these technologies' context, they often say little about the specifics of their operation. Access to

*Both authors contributed equally to this research.

Authors' addresses: Daniel Cardoso-Llach, Computational Design Laboratory, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA, 15217, USA, dcardoso@cmu.edu; Eric Kaltman, Computer Science, California State University Channel Islands, 1 University Drive, Camarillo, California, 93012, USA, eric.kaltman@csuci.edu; Emek Erdolu, Computational Design Laboratory, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, USA, eerdlou@andrew.cmu.edu; Zachary Furste, Carnegie Mellon University Libraries, 5000 Forbes Avenue, Pittsburgh, Pennsylvania, USA, furste@cmu.edu.

This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

© 2021 Copyright held by the owner/author(s).
2573-0142/2021/10-ART294. <https://doi.org/10.1145/3476035>

functioning software systems as primary sources is thus essential. With this as a premise, we investigate the potential of distributed emulation networks to support research and pedagogy into historical and sociotechnical aspects of software. We document our use of the Emulation as a Service Infrastructure (EaaS) distributed emulation network [37] in the context of a university course on Computer-Aided Design (CAD). To elucidate the potential of distributed emulation in a design-pedagogy context, we describe the course participants' use of the EaaS system to examine historical versions of AutoCAD, a widely-used commercial CAD software.

The paper's main contribution is to offer a detailed case study of the implementation of distributed emulation networks as a resource in software research and design pedagogy contexts. This is demonstrated in the methods and analyses presented, which use emulation to access historical versions of Autodesk's AutoCAD, and to distill its core functional and sociotechnical propositions. Through comparative reflection between different versions of the software, certain design observations and critiques become possible due to the emulated interaction that would likely not be derived from other historical sources. An additional contribution is to review historical and socio-technical engagements with software systems from across Computer Supported Cooperative Work (CSCW), Human Computer Interaction (HCI), and Science and Technology Studies (STS) including recent works exploring visual and tactile aspects of historical systems through a variety of methods, including some early uses of emulation as a means for making software interactions available as primary research materials.

In contrast with other scholarly methods investigating software and software-based practices, emulation takes advantage of existing software-archival resources, and opens up historiographical opportunities by, for example, giving direct access to a system's interfaces, functional characteristics, and workflows. However, emulation has received relatively little attention as a vehicle of scholarly research and, while previous work has addressed the design of emulation interfaces for understanding emulated systems [54], there are no prior examples or detailed case studies of its use in a design pedagogy context. In our study, distributed emulation was incorporated into the course as a resource in the development of speculative computational design systems. It is part of an ongoing collaboration at Carnegie Mellon University between faculty and students in the School of Architecture and the University Libraries, and is organized around a course on CAD.

The paper is structured as follows: The Background section reviews work in CSCW and adjacent fields engaging with historical questions and methods. Of interest here is the growing body of research that expands beyond the usual toolkits of CSCW and historical research to engage with visual, material, and interactive dimensions of historical software systems. Noting the relative absence of emulation as a research method in design and pedagogy contexts, emulation is described along with examples of its use in historical software research, and a background of AutoCAD and its history is offered to highlight its important role as an industry standard in shaping architectural and engineering design practices. The Methods section presents the course and its pedagogy in greater detail, situating emulation along with reconstruction and speculation as distinct but complementary methodological approaches into software. Further, it illustrates EaaS's system architecture, configuration, and use within the context of the course. The Results section presents a synthetic overview of the work of the class, with a focus on the use of emulation to analyze two different versions of AutoCAD: AutoCAD R12, released in 1992, and AutoCAD 2000, released in 1999. This section shows how the use of emulation, in conjunction with other research methods, allowed for close, user-centered analyses of the software's interfaces. These analyses opened paths for comparative reflection that revealed aspects of the software's evolution that would have been otherwise difficult to identify. Here we find similarities between the kind of software analysis enabled by emulations and the analytical technique of close reading, common in interpretive analysis of texts in the humanities. In addition, the section shows how the analytical and interpretive work

conducted through the emulations helped participants develop design concepts for two speculative design tools: a tangible interface for digital drafting, and an assistive, eye and voice-activated, CAD system. The Discussion section distills from the analyses a series of methodological, pedagogical, and design insights. Finally, offering a reflective overview of the research, the Conclusion articulates emulation as part of an expanded toolkit for software research and pedagogy, and outlines this method's strengths and limitations.

2 BACKGROUND

2.1 Software as primary research material: Historical perspectives in CSCW and HCI

As noted in [48], historical perspectives and methods can contribute to studies of computer-supported cooperative work by helping to develop a deeper understanding of the social, material, and epistemic aspects of technological systems, and opening new avenues for inquiry, speculation, and design. And yet, work leveraging these methods remains relatively sparse in the field. Exceptions include efforts to, for example, trace the evolution of computer interfaces from their 1950s origins in academia and industrial labs into end-user products in contemporary settings [23] and the historical development of interaction techniques — symbolic manipulation, the mouse, windows, text editing, hypertext, gesture recognition — in universities and research labs since the 1950s [36]. Other research has usefully explored beyond prominent institutions and pioneers, focusing instead on the “mundane mainstream,” or the “mainstream development of mainstream user interfaces to mainstream software used in mainstream organizations” [26]. Other researchers have worked to articulate technologies as historical artifacts whose design is fundamentally informed by past systems [7], and engaged extensively with documentary and oral sources to understand their context [55] in order to draw insight or open up pathways for design. A related, interpretive strand of research has sought to situate technological infrastructures historically to highlight their shaping of bureaucracies, professional practices, and public understandings [9]; to clarify the relationship between information infrastructures and the knowledge practices they support [41]; and to critically trace their ideological commitments and their linkages to emerging forms of pedagogy, activism, and labor [3, 29]. Our use of emulations to study the historical evolution of a CAD system widely used in the architecture and engineering fields shares aims with this interpretive tradition, and with the “mundane mainstream” approach described in [26].

These trends show how historical questions and methods have entered CSCW and adjacent fields in the service of documentation, reflection, and critique, and contributed to both scholarly and design questions. However, the examples reviewed so far have mostly relied on the conventional tools of historical research — oral, archival, and other documentary sources — which are invaluable to illuminate technologies' contexts, but are not always well suited to examine their specific dispositions and modes of operation. If we are to understand software systems' role in shaping the practices and lived experiences of their users, as well as their material effects, we need to consider them primary research materials. This poses challenges as historical technological artifacts depend on conditions that can be difficult, or impossible, to reproduce. A recent strand of research at the intersection of CSCW, HCI, STS, and design offers elements to address this question by employing an expanded methodological repertoire to engage with visual, material, and tactile aspects of historical technological artifacts.

For example, HCI researcher and designer Daniela Rosner has combined oral history and archival research into participatory design experiences to critically re-stage the process of making magnetic-core memory, helping reveal the forgotten “gendered forms of craftwork” that underlie the earliest computers [43]. Seeking to explore the crucial role of analog systems in the history of scientific documentation, STS scholar Yana Boeva and co-authors have reconstructed the logic of historical

hardware artifacts using modern programming languages [8]. Drawing from archival materials and oral sources, one of this article's authors has reconstructed early CAD systems using modern programming languages and hardware, making aspects of their logical, visual, and gestural design available for both historical and design inquiry [13]. Related work has sought to collect digital traces produced during software-based design processes and analyze them as historical and ethnographic materials [12, 18, 56]. Work in this space has shown potential to enrich our understanding of historical systems, and of the collective and distributed nature of design processes enabled by software-based modes of collaboration.

However, while software or hardware reconstructions can usefully enliven the logic of systems that depend on long-lost hardware and software, they can be impractical when dealing with systems of a certain level of complexity. In addition, many software systems developed during the latter part of the twentieth century, along with their respective operating systems, are increasingly available in institutional archives and collections, making their reconstruction redundant. While trace ethnography and software probes might be particularly useful in sociotechnical studies of present-day sites, they offer little insight into historical software systems and software-based practices. It is here that emulation can be an effective vehicle of research into software and design.

In contrast with the approaches reviewed above, emulation takes advantage of existing software-archival resources. Software collections in libraries and other institutional archives increasingly account for the aforementioned “mundane mainstream” [26] of software: widely used systems which are not conventionally regarded as innovative. By offering access to these systems, emulation can help shift the focus away from emblematic and well studied systems, placing it instead on software systems that have received less scholarly attention but which constitute — as a result of their market footprint and other factors — important sociotechnical infrastructures. The following section reviews the technical paradigm of emulation and of some of its historiographic affordances.

2.2 Emulation: Definition and potentials for historical research in CSCW

In computer science and adjacent fields “emulation” generally describes any method or system in which a specific computational configuration is imitating or approximating a different one. In order to provide definitional and conceptual clarity, this paper borrows the more specific use of “emulation” from the software preservation and archival communities to distinguish it from a raft of other uses in computational hardware and network design. In particular, we adopt Rosenthal's terminology that defines “emulation” and “virtualization” in the specific context of library collection-based distributed emulation solutions [42]. As a result, our use of “emulation” is an umbrella term referring to “emulation or virtualization depending on the overlap between [virtual machine] and host instruction sets.” A “virtual machine” (VM) in this definition is a collection of software that mimics the behavior of some physical machine from within a containing host computer system. In general, the “physical machine” mimicked by a VM can either share an instruction set architecture (ISA) — the specific set of processor specific machine instructions that executable software executes — with the host machine or use a completely different one. In most definitional schemes, “virtualization” applies to a VM and host sharing the same ISA (or a subset) whereas “emulation” applies to situations where the ISAs are different [46]. An “emulator” is, therefore, a specific VM that allows for unmodified legacy software to run on a host system that was not designed or architected for that system. While this elides some important technical distinctions between virtualization concepts, the purpose of emulation in this work is to provide access to legacy software on modern hardware without the need to deeply comprehend these distinctions. Accordingly, we use the term “emulation” rather liberally throughout this paper, with some specific software programs referred to as “emulators” even though that term slightly miscategorizes their full feature set.

Emulation has an important potential as a historical methodology in CSCW. Researchers in this field are concerned with digital images, chat-room transcripts, e-mails, computer-aided design drawings, data sets, and other digital traces. In addition, they consider the specific means through which specific hardware and software systems condition past ways of working [44]. Through the lens of preservation, emulation appears to provide an essential function in allowing future access to these and other forms of technologically-specific — and dependent — documentation [22, 45]. While historiographic uses of emulation are sparse, there are precedents for emulation as an interpretive historical methodology. For example, researchers have taken advantage of emulators to restore and analyze past systems, such as Fortran [34], the Arpanet Interface Message Processor (IMP) [49], the original EDSAC [10] — to reproduce the first instance of “debugging” — and Tesselator — an object-oriented graphics system in Sweden from the 1980s [32]. Video game studies has also made use of embedded emulation in web pages to provide historical context [28, 35]. Of particular relevance to this study, the FACADE project at MIT used emulation to aid in the digital archiving of CAD files from architectural projects [47].

From a historiographic perspective, emulation affords access to software as a primary, interactive source. Since more and more historical records will be mediated through software use, as noted in [38], “the future of sound historical interpretation is going to be dependent on cultural histories of [software] platforms.” In contrast to recorded audiovisual media [15, 19, 52], emulated software affords access to a user’s “procedural interaction” with a systems [24]. By making recovering interactive experiences accessible, emulated software offers additional forms of context that contrasts not only with verbal descriptions, but also with image- or video-based screen captures of software execution. As we show below through emulation-based analyses of computer-aided design systems, running legacy software environments has the potential to allow for novel encounters between users and historical modes of interaction.

2.3 AutoCAD: Outline of an industry standard

Computer-aided design programs are professional design tools which help to model the physical configuration of many built artifacts. Moreover, their interfaces and modes of operation play a role in shaping the intellectual and material labors of entire professional groups and in structuring cooperative work in architecture, engineering and other design fields. In the interfaces of mainstream software systems such as AutoCAD, we may thus find important clues to deepen our understanding of modern architectural and engineering work. Our focus on an industry standard such as AutoCAD nods to the call to re-focus software studies on the “mundane mainstream” [26]. Typically overlooked by scholarship, these systems shape the intellectual and material labors of entire professional groups — engineers and architects in this case.

It is in fact hard to overstate AutoCAD’s significance in the history of CAD systems, or its impact on architecture and engineering practices. First released in 1982, AutoCAD was the first commercially available CAD system for personal computers, as opposed to mainframe machines, and the first to gain a significant market footprint [16]. The proprietary file format developed by AutoCAD, .dwg, remains, to this day, an industry standard. At its core, AutoCAD is a graphics program created by one of the companies’ founders, Michael Riddle, in 1977, called “Interact.” The first Control Program/Monitor (CP/M) and IBM PC editions of AutoCAD were versions of Interact ported to the C programming language [51]. It was named AutoCAD-80 because it was designed to run on machines with a Zilog Z80 or Intel 8080 processor, and gained popularity at COMDEX, a popular computer trade show of the time in Las Vegas, in November 1982 [50]. AutoCAD’s commercial success came a bit later, linked to the rise of relatively affordable commercial desktop systems during the 1980s. Sales exploded in 1984 with the release of AutoCAD-86, which ran on Intel 8086, Victor 9000, and IBM PC machines [51]. Equipped with a mouse and a keyboard, these

machines facilitated a host of new interaction techniques as well as windows-based graphical user interfaces [36]. Taking advantage of these developments, and of the relative affordability of personal computers, AutoCAD became the first commercially successful CAD system, and helped kickstart what would become a multi-billion industry in the US [14]. Early advertisements for the system emphasized key selling points, such as its affordability and its capacity to run on a desktop workstation (Figure 1). By 1984, the software alone sold for 2,000 USD and, together with a workstation, for close to 15,000 USD [4].

The earliest versions of the software crucially featured a drawing “space” for creating and editing geometric shapes such as points, lines, and polygons. Through entering text commands, activating screen icons, or pressing on tablet-based inputs, users could control a variety of drafting operations. This flexibility of input was advertised as a convenience. It made the system easier to learn and use. AutoCAD’s key proposition was to computerize traditional drafting operations. Its interface offered users an “Euclidian environment” allowing them to configure geometric elements with relative ease while offering practical advantages for design documentation [11]. Making lines, moving them around, copying them or deleting them, overlaying drawings, and “saving” them was different in AutoCAD than on traditional, ink-on-paper drawings. Digitization promised faster and more cost-effective design processes, and heralded important shifts in architectural and engineering practices. The evolving interfaces of CAD systems, and of AutoCAD in particular, staged these shifts in ways that are worthy of analysis.

How our \$2,000 CAD software can keep you from burning up your design budget.

Until now, high-quality, professional computer-aided design could only be done on expensive, large-scale mini or mainframe computers. But now there's AutoCAD. A full-function graphics program that turns your low-cost personal computer into a full-fledged CAD workstation. So that designing an AUTOMATED 50-350 GPM-class fire nozzle has never been easier. Or more cost-effective.

90% of mainframe CAD at 5% of the price. AutoCAD is designed especially to work on today's most popular microcomputers to fully automate the drawing process. For just about any drawing job you can think of. With its extensive, push-button editing facilities, AutoCAD lets you DRAW an object (or any part of one); ROTATE or SCALE it; DRAG it; ZOOM in and out on it; STORE it away in your own parts library and call it up whenever you need it; FILL an area, use an unlimited number of LAYERS; even automatically DIMENSION distances and angles in any part of your drawing. And, at only \$2,000 for the complete AutoCAD program, you can have all this power in a fully-configured, top-of-the-line desktop workstation for less than \$15,000.

You'll learn it in two days. The beauty of AutoCAD is that you don't have to know a thing about computers. Even if you've never touched a keyboard, you'll pick it up in a matter of days, and feel comfortable inside a week. Sound too good to be true? A simple-one-touch command structure always points you in the right direction. A HELP command is always available to keep you on track. And AutoCAD even lets you choose from a growing number of manufacturers.)

PC boards, skyscrapers, and everything in between. No matter what type of drafting your work involves—from electromechanical circuit layout to structural schematics to architectural design—AutoCAD can give you a whole new perspective on the drawing process. And on saving money. Because whether you're in a two-person shop or a two-billion dollar company, you'll find that the AutoCAD system pays for itself in just a few short months. Beyond that, your AutoCAD software will run on newer, and more powerful computers when they become available—so that you can be sure your investment in creating drawings and training staff will never go up in smoke.

Fire off a letter. There are currently over 7000 AutoCAD users worldwide. If you'd like more information, write or call your nearest AutoCAD dealer. He or she can provide you with all the details, including literature, specifications, and hands-on demonstrations. AutoCAD. For designers, it's why the personal computer was invented.

Zoom in on a part to focus on the details. AutoCAD automatically dimensions distances and angles.

Would the Tumble be better where the stove is? Use the MOVE command from among a wide variety of input devices to suit your particular needs: including pointing devices, mice and easy-to-use electronic drawing

Architectural layout courtesy of Learning Path, Architects. Solar Panel Assembly plot courtesy of Future Technology, Inc.

AUTOCAD
AUTODESK, INC.
350 BROADWAY
SAN RAFAEL, CA 94905
TEL 415/331-0955
TELEX 756521 AUTOCAD UD

Fig. 1. AutoCAD ad in Scientific American Magazine from 1984 [4]

3 METHODS

3.1 Course overview

We introduced the EaaSI system as a pedagogical resource in one module of a graduate level course on CAD. An elective seminar, the course tasked students with analyzing and interpreting the history of specific CAD systems through the use of three conceptual methodologies, which were explored through the sequence of course modules: “reconstruction”, “emulation” (the focus of this paper), and “speculation.” Software reconstruction is the use of modern computational production techniques to approximate past software systems and their functionality. An interpretive and reflective remediation technique that requires both technical skill and detailed documentary research, reconstruction aims to illuminate past interaction paradigms when there is no direct access to the original system. This technique invites a practitioner, in this case a student, to grasp a historical system’s salient visual, tactile, and interactive aspects in order to approximate them using modern techniques, thus making key aspects of the system available for analysis [13]. During the reconstruction module students researched a specific historical CAD system, re-implemented it, and gave a research presentation covering 1) its historical context of use, 2) the function of the software within its specific collaborative domain, and 3) an analysis of the visual and interactive aesthetics of its interface.

The emulation component of the course, conducted in collaboration with the University Library and the EaaSI project team, and the focus of this paper, tasked students with choosing an available CAD system on the university’s EaaSI node and evaluating it through extended collaborative use in the EaaSI browser interface. Similar to the previous module, students were invited to research their CAD system’s historical and sociotechnical context, as well as its visual and interactive qualities. Given that the emulated software environments provided direct access to the historical systems’ interfaces, students were able to directly evaluate different versions of historical systems, and compare them with their contemporary counterparts. Instead of relying on documentation to interpret and analyze a legacy system, students were able to engage with their chosen software as a primary research material to aid explicit use and tacit comprehension. Students had two weeks to familiarize themselves with the EaaSI interface and investigate their chosen CAD systems.

The speculation component of the course invited students to synthesize their experiences with reconstruction and emulation into the design of a novel, experimental CAD system. Drawing from the pedagogical tradition of the design studio, this section of the course developed through iterative presentations and critiques of the work by faculty and peers. Each experimental CAD prototype thus progressed into a demo stage suitable for live interaction in an open, final presentation. In creating experimental CAD systems, the course sought to mobilize the historical understanding of the software towards generative outcomes.

3.2 Access to the emulations framework

The Emulation as a Service Infrastructure (EaaSI) project is a distributed emulation platform that allows students and researchers at partner institutions to share and access pre-configured, cloud-based emulations of legacy computer environments through standard web browsers [37]. Like other preservation-minded emulation systems, EaaSI arose from the library and archival communities’ concerns regarding access to legacy software objects accumulating in institutional collections [21].

EaaSI’s architecture is organized as a distributed set of microservices that manage the configuration and execution of cloud-based emulations (running in virtualized containers) whose output is displayed in a standard HTML5 Canvas element on a user facing web page. Users can manipulate the remote emulation as the EaaSI webpage will forward user triggered mouse and keyboard commands across the network. The EaaSI implementation is the first multi-institutional distributed

use of an EaaS system. Previously, EaaSI’s internal EaaS components developed from singular use cases, such as sharing CD-ROM collections [5] and as a virtual library “reading room” for software collections [40].

Each member of the EaaSI network runs an EaaSI “node”, a web server that manages each institution’s collection of pre-configured environments. In our use case, “base” Microsoft Windows XP and MS-DOS environments installed in instances of the QEMU emulator [6] were used to support the further installation of derived “software environments” containing dependent CAD systems. This allowed for a set of different CAD environments to be available for student use on our node. Although EaaSI provides a browser-based UI to manage the setup and provisioning of emulation environments, there is still a significant amount of work required to import and install legacy software. Once the specific software package is located, it needs to be imaged from physical media into a compatible disc image format, mounted into the correct base environment and installed according to the requirements of the time of its production. In the case of expensive commercial software this means dealing with potential digital rights management schemes that may no longer work, or specific driver or dependent software requirements that necessitate another search and installation process of their own.

Prior to the beginning of the Emulations module, a collection of several historical CAD software systems were prepared in the EaaSI framework for students to explore and choose from. These were installed directly from physical media collected at the University Library and School of Architecture archives. Using Guymager [2] within the BitCurator [33] toolkit, we converted the images to .iso format using the dd command line utility [1] and uploaded them through the EaaSI interface. In the case of AutoCAD R12, we cloned a software environment shared by Yale University Library. Some of the software required dependencies, such as unzipping utilities, for installation, while others required troubleshooting operating system parameters.

Fig. 2. The EaaSI interface for selecting object environments (left) and software environments (right). “Object” environments are configured for specific files or collections of files. Right clicking allows for “Run Environment” which leads to the screen in Figure 3.

After the software environments were configured on the EaaSI node, students were each given a login and password to enter the administration interface and startup their chosen environment. Figures 2 and 3, respectively, show the selection and emulation interaction screens presented by the EaaSI UI. The emulation interaction screen is fairly basic, providing the ability to start and stop the emulation, save out a derivative environment if necessary, change media — to load different objects — and send ESC and CTRL/ALT/DELETE keyboard signals. Mouse and keyboard input

Fig. 3. EaaS UI with a running instance of ArchiCAD 9. This is the interface through which student interaction was conducted.

are captured by the HTML element holding the emulation component's output from the EaaSI server. Input latency in our use case was rather low due to both the students and EaaSI server being located locally on campus.

3.3 Details, scope, and analytical emphasis

The students in the course had different disciplinary backgrounds including computer science, architecture, engineering, art, and design and included undergraduate (3), masters (1), and doctoral (2) students. Their degree of skill with CAD software was similarly varied, with the architects' level of expertise being the highest. In the reconstruction module, students worked to reconstruct systems including William J. Porter's 1968 urban planning system DISCOURSE [39], the 1971 drawing robot AARON by Harold Cohen [17], and Shape Grammar Interpreter, a rule-based CAD system developed in 1986 by Ramesh Krishnamurti [30]. In the emulation module, students emulated software systems including VistaPro, an early generative landscape design software, Maya 2010, a NURBS-based 3-D modeling and animation software, and different versions of the architecture and engineering software AutoCAD.

Rather than giving a detailed account of the work produced by the whole class, the Results section below focuses on the doctoral students' analyses of two versions of AutoCAD running on the EaaSI emulation framework, and then briefly describes the speculative systems the emulations helped inspire. One analysis focused on the DOS-based AutoCAD R12, released in 1992, and the other on the Windows-based AutoCAD 2000, released in 1999. Focusing on these two analyses allows us to highlight what we consider the key analytical potentials of emulation for software research and pedagogy. First, by affording access to different versions of the same software system,

they show how emulations can support new understandings of the historical evolution of software systems. Second, focusing on two examples allows us to document the analyses in more depth, enabling us to show how distributed emulations facilitated what we term here "close readings" of software. Close reading is a technique used in the humanities to identify the features of a written text via patient, systematic observation and analysis [27]. The section is followed by a discussion section that reflects on the readings and on the potential they provide in enriching conventional accounts of software based on archival or documentary evidence with performative ones. These allow for a deeper understanding, and raise new questions, about software's functionality and interactivity.

4 RESULTS

4.1 Close reading of AutoCAD R12 (1992)

4.1.1 Emulation. Josie, one of the students in the class, is a professional architect, expert AutoCAD user, and graduate student in architecture. Trained in the 2010s, and with a few years of industry experience in architectural firms, he is most familiar with more recent versions of the software. He calls, for example, the 2006 version of this software the "classic." He recalls being first exposed to AutoCAD during his 2nd year in architecture school:

I opened AutoCAD and learned six commands: line, erase, trim, copy, move, and zoom. After a while, I realized that these six commands were the most essential commands in AutoCAD for [making] drawings (...)

He further recalls the particular excitement when his dexterity with the software afforded him a measure of aesthetic satisfaction. For him, lines drawn in AutoCAD had "[u]nique blade-like clarity, bleeding edge accuracy, lines that seem to be hard but contain the user's drafting identity, and information that appears to be mixed in a complex but subtle way." Once he learned to use the icons on the Windows-based GUI, he "forgot the existence of the command bar for about 10 years." However, these days are gone. Josie considers himself an expert user. Instead of the icons, he is adept at combining text commands — swiftly typed with his left hand — with the precise motion of the mouse under his right hand to point and select elements on the screen. He feels that newer versions of AutoCAD have made the software too heavy, and too cumbersome. In fact, he has customized his version of AutoCAD 2018 to resemble AutoCAD's "classic" version. The software currently it's trying to do "too much," he says. There is a certain purism at play.

For the course's emulation section, Josie chose to explore AutoCAD's 1992 release, the DOS-based AutoCAD R12. Running the software on the EaaSI distributed emulation platform, he set out to use it and compare it with the modern versions of the software he knows. What would the experience of using a relatively archaic version of AutoCAD, running on DOS, reveal for him?

Using R12 confronted Josie with what seemed to be a purer, more elemental version of the system — something closer to its "essence." To test the system, he drew the floor plan of a simple house (Figure 4, left), and reflected on his experience. To his surprise, while certainly more basic, R12 offered a familiar environment. The two key elements of AutoCAD's interface, the command line and the drawing "space," were simpler and more deprived in the DOS-based R12 version than in the more recent, Windows-based, versions he knew. But they were there as a reassuring presence. In particular, being able to use the command bar to enter AutoCAD's traditional drawing commands — line, offset, trim, etc — was comforting: "the presence of the fixed command bar in R12 [offered] great stability in the usage of an unfamiliar software" he recalls.

The presence of layers was also a reassuring feature. Layers are a key feature in most CAD systems. They allow users to overlay drawings on top of each other — a sort of digital version of the yellow trace paper traditionally used by architects. Layers can be conveniently turned on

Fig. 4. Josie's drawing (left) and the layer settings panel (right) in AutoCAD R12, both via EaaSI

or off to, for example, hide some details of the drawing in order to focus on a certain aspect of a design. In Josie's test drawing, for instance, the construction lines were placed in one layer, the section lines in another, and the elevation lines in a third. In recent versions of AutoCAD, layers are managed in a pop-up panel containing a list of the layers in the drawing, which affords users detailed control over the characteristics of each layer, including color, name, editability status, and printing properties among others. By contrast, in R12, after typing the "la" command no panel popped up. Instead, the command line returned a dialogue with a list of options:

Layer?/Make/Set/New/ON/OFF/Ltype/Freeze/ Thaw/Lock/Unlock.

He was thus able to create a new layer by typing "New" and the name of a new layer. Later Josie realized that there is a "layer" function in the screen's top menu which activates a panel that activates a list, as well as visual access to these options (Figure 4, right).

Finding these and other alignments between two versions of a software almost 30 years apart created a sense of familiarity. Accessing a historic version of AutoCAD via the emulation gave Josie access to the system's skeletal structure. Of relevance here is AutoCAD's concept of interaction: the mapping of analog tools — rulers, T-squares, ink-pens, tracing paper — into a digital interface, and its persistence across multiple versions and operating systems (Figure 5). More specifically, in the transit between AutoCAD R12 and 2000 (and between DOS and Windows) we see a familiar move from textual and symbolic information towards a skeuomorphic interface where the mapping between analog tools and digital ones relies on increasingly literal figurations. The "line" function, triggered by the "l" text command or the word "line" on the screen of AutoCAD R12, is, in addition, re-presented as the drawing of a line on a beveled button on the screen of AutoCAD 2000. Similarly, the "dimension" function, triggered by the "dim" text command on the word "Dim" on the screen of AutoCAD R12 is, in addition, re-presented as the drawing of a ruler on another button on the screen of AutoCAD 2000.

The comparison speaks to the technical evolution of the interface, but also of the relative stability of AutoCAD's core functions and concepts of interaction. In Josie's words:

[During the emulation] I became aware of the consistency of AutoCAD's interface, its direction. The fact that users who have never used an older version of AutoCAD can create a drawing without [too much difficulty]... shows that the software's principles of operation, its interface... have a certain consistency. This consistency can simply be interpreted as a result of AutoCAD's conservatism — but perhaps this is the orientation that AutoCAD has had since its inception.

Fig. 5. A mapping of analog drafting tools — rulers, T-squares, ink-pens, tracing paper — into the digital interface of AutoCAD R12 via EaaSI.

Released in 1992, AutoCAD R12 interface was based on MS-DOS (Figure 6, left). With the release of R13 a year later, the interface of AutoCAD adopted the visual style of Microsoft Windows (Figure 6, right). R12 was, in a way, the last of a kind.

Fig. 6. AutoCAD R12 interface via EaaSI (left) and AutoCAD R13 interface (right) (<https://dwgacad.blogspot.com/2009/12/autocad-r13-november-1994.html>)

4.1.2 Speculation: A voice and gaze activated CAD system. Through the use of emulations, Josie was able to observe and reflect upon the consistency of two of AutoCAD’s key interface elements — the command bar and the coordinate input system — across different versions. This highlighted for Josie the software’s core proposition and values, which he described as a digital version of a drafting table. An experienced draftsman himself, and thus familiar with the repetitive nature of drafting in the architectural workplace, he started to wonder about more efficient ways of calling commands and entering coordinates. Drawing from his analysis of AutoCAD R12, he started developing ideas for an alternative input system to replace them (Figure 7). Could a CAD system be controlled more intuitively via a user’s gaze and voice?

Fig. 7. Josie's sketches of the prototype showing voice- and gaze- activated control

Instead of entering a full-name, shortcut, alias of commands, Josie imagined, users could say a command's name and respond to the system's prompts for more information (Figure 8, left). He created a quick prototype using Antoine Lamé's gaze tracking library [31] and Google's speech-to-text voice recognition model [20]. The user's gaze plays the role of mouse pointer, and blinking patterns work like a variety of mouse clicks (Figure 8, right). For example, with one blink users can tell the system to "drag" elements on the screen. With two blinks, a command is confirmed. The prototype highlighted some interesting problems. While Josie thought that the gaze-pointing feature would entail increased control, it actually proved to be counter-intuitive:

Aside from [technical problems related to the] immaturity of implementation, the prototype confirms what now seems like a very obvious proposition: that seeing is not drawing. The eye has priority as a "sensor" to receive information. Directly and immediately linking viewing to the act of drawing doesn't work very (...). In other words, another step between viewing and drawing should be like a buffer.

The prototype allowed Josie to analyze the potentials and challenges of these new ways of input, and to realize that "seeing is not drawing." In other words, that seeing and drawing are different and mutually supportive activities. By attempting to collapse them into one, neither was possible. This realization led Josie to think that not everything needs to, or should be, made faster, or more automatic.

Fig. 8. Drawing a rectangle with voice activation (left) and voice plus gaze interaction with CAD (right)

4.2 Close Reading AutoCAD 2000 (1999)

4.2.1 Emulation. Jer is an architect and urban designer, and a graduate student in architecture. Jer is also a proficient AutoCAD user. He learned it during his architectural training in the early 2000s and, like Josie, perfected his skills while on the job at several architectural firms. For the emulations study, Jer chose to examine AutoCAD 2000. He saw this as an opportunity to approach analytically, and with critical distance, the tool he was introduced to when he was a young student, and which he knew mostly as an instrument of production:

I deliberately chose AutoCAD 2000 as I wanted to see how the user interface has changed in graphic style over the last twenty years. I have seen every single version since, so I was curious to see what was new in the 2000 version, and what from this version has remained with us.

Jer explored version 2000 on the EaaSI platform, using architectural drawings found in the platform to interact with the software and examine its graphic user interface (GUI). To Jer, AutoCAD 2000 seemed starkly simple in comparison with more recent versions. The number of interactive icons and operations was “manageable,” the interface was decluttered, which underlined for him some of the system’s basic categories of user functions. For example, file management operations included saving and copying drawings, managing layers, as well as associated files (Figure 9). Starting with the R11 version, AutoCAD offered users the possibility of managing complex drawings by linking multiple, externally referenced files, or “x-refs.” The ability to reference external files is crucial as it made it easier for architectural organizations to subdivide and manage design work across large teams.

Fig. 9. File management, screen management, and object manipulation command sets in AutoCAD 2000

Another distinct category of functions concerns screen management operations, and includes commands for navigating the screen such as zoom, pan, or orbit which allow users to interact with the drawing space (Figure 9). These highlighted for Jer some of the translation work between analog and digital happening in AutoCAD’s interface. When working on paper, architects must calibrate a drawing’s level of detail to the paper’s size. AutoCAD’s virtually limitless “cartesian” drawing space, and the ability to freely zoom in and out, and “orbit” the design from any angle, releases architects from the constraints of paper, enabling them to work at full-scale all the time. Finally, the object manipulation panel contains commands for creating, manipulating, and deleting geometric objects such as lines, polygons, and arcs (Figure 9). While working in AutoCAD 2000’s

drawing “space,” Jer found himself dealing with shapes in strict procedural fashion. For instance, creating a mirror image of a triangle involved calling the “mirror” command by entering “m” in the keyboard, using the mouse to select the triangle, specifying a symmetry axis by clicking on two points on the screen, and confirming whether to delete the source object or not.

Fig. 10. Copying a circle via the command bar and mouse in AutoCAD 2000 on EaaSI (left) and copying a circle via Visual LISP code (right)

Jer learned that aside from command-line input the earlier, DOS-based versions of AutoCAD offered users the option to click directly on interactive text commands on a screen menu (Figure 6, left). Windows-based versions of AutoCAD, starting with the software’s thirteenth release in 1994, feature a graphical user interface with pictorial icons that resembled the tools’ analog counterparts (Figure 6, right). In addition, since AutoCAD 2.1, released in 1986, users can create custom programs in AutoLISP, a programming language based on the LISP, to automate drawing routines. This functionality was enhanced with the introduction of the Visual LISP IDE in 1999. (Figure 10, right).

To Jer’s surprise “the fundamental operations of the software and the objects of interaction have not changed very much since 2000.” However, he wondered about the implications new features had on architectural design work. For example, he looked closely at the Layer and Plot Settings functions (Figure 11). AutoCAD 2000 was the first version to introduce a “multiple layout” mode for previewing various plot styles, and the possibility of copying linetype and layer settings across drawings. Reflecting on his own experience as an architect in practice, Jer pondered what these changes might have entailed for architectural design work: “These features were central for establishing custom standards for layering and documentation which is common practice in architectural offices, including the ones I worked at.”

Another feature introduced in AutoCAD 2000 was the multi-document interface, which allowed users to manage other drawings as external references to the “master drawing.” This offered architectural organizations the possibility of distributing design work across large teams. With a “master drawing” managing a large number of associated files in separate files and computers, different designers or teams could work on specific aspects of a design.

Computer work also inscribes changes to bodily postures and gestures. Working with the emulations Jer was able to observe how, in two decades of evolution of AutoCAD’s interfaces, these

Fig. 11. Layer properties (left) and plot styles (right) in AutoCAD 2000 via EaaSI

interactions had barely changed. Drawing from pre-digital drafting practices, he wondered about the bodily dimension of design work, and how standard input devices, keyboard and mouse, shape and constrain them (Figure 12). With this, Jer imagined a speculative device to experiment with other motions and physical qualities in design work.

Fig. 12. Drafting in the 1980s (left) and a view from a contemporary architecture office (right) (<https://www.vintag.es/2018/08/life-before-autocad.html>) and (<https://www.designboom.com/architecture/bjarke-ingers-group-big-architects-studio-visit-01-08-2014/>)

4.2.2 Speculation: A Tangible CAD Interface. Like Josie's, Jer's speculative CAD interface sought to avoid the conventional text-entry command input. Instead of voice and gaze activation, however, it relied on a custom tangible interface which replaced keyboard input for entering commands. The device comprised a surface with four sensors, and four sliding levers. With these, users could create different configurations of commands. For example, they could align with the six most commonly used commands in the software (Figure 13). With these elements, Jer sought to depart from conventional input devices and create an experimental CAD interface engaging with users through a different gestural vocabulary.

By sliding or moving these objects on the surface, a user can block or expose the sensors, and create the intended configuration to activate the corresponding command (Figure 14). The operation

Fig. 13. Diagram of interaction displaying how the six CAD commands are activated with Jer's interface

Fig. 14. Elements of Jer's speculative interface

is executed with the mouse by using left click to interact with the drawing space and right click to enter the finished steps of a command (Figure 14). This inchoate exploration allowed Jer to see a new set of gestural possibilities for computer-aided design.

5 DISCUSSION

5.1 Consistency and familiarity

The close readings documented above show how, despite its nearly four decades of existence, some functions remain relatively constant across AutoCAD's different versions. As Josie noted for the Layer function, and Jer noted for the Mirror function, many of the system's operations haven't fundamentally changed since earlier versions. For example, to copy a geometric object, users type "copy" (or simply "c") on the software's command bar, which prompts the user for additional information to further specify the operation. What object needs to be copied? What should be a reference point for the copy? Where should the copy go? The combination of mouse and keyboard input was the same throughout the versions observed regardless of the implementation and presentation of the graphical user interface (Figure 10, left).

This observed consistency elicited different but related observations from both students. Josie attributed it, critically, to the “conservative” character of the software, but also expressed a sense of familiarity and delight when he discovered that he could fluently use the tool. Also critically, Jer described the newer versions of the software as “cluttered.” In spite of these critical elements, in their readings we saw what could be construed as a nostalgic attachment to familiar, purer versions of the system, and a rejection of the profusion of tools, toolbars, and icons in more recent versions. Interestingly, their analyses both identify and document features and patterns of interaction that, remaining unchanged throughout the software’s evolution, can be seen to configure a distinct functional and interactive proposition that is separate from the specific implementations of AutoCAD’s graphical user interfaces.

Jer combined his observations about the software’s usability with a keen curiosity about the impact of new features on the organization of the architectural workplace. He wondered about the organizational implications of specific features introduced in later versions of the software, such as layers and plot styles. This line of analysis usefully hints at the possibility of further research into how interfaces might be agents of organizational change both within a firm and within broader professional groups. This type of analysis aligns with the growing body of CSCW literature that are concerned with reflection and critique of technologies, as discussed in Section 2. By contrast, Josie’s account of usability is more personal, and offers a glimpse of the intimacies users can develop with, and within, software interfaces. As noted, the system’s varied forms of input foster users to develop equally varied forms of fluency. A user’s specific habits of clicking, typing, or coding are deeply personal while at the same time indicating a degree of proficiency. For example, as Josie’s account makes clear, experienced AutoCAD users rarely use the GUI icons, and take pride instead in their facility with the command-line. Users able to write customized AutoLISP scripts to automate aspects of drafting are rare, but can be seen as the software’s most proficient users, able to engage with it at a lower level of abstraction.

5.2 An Archive of Interfaces

The materials assembled so far illustrate how in a design and pedagogy context emulations can supplement archival and documentary sources by giving access to visual, functional, and interactive aspects of software. Rich in images and detailed descriptions, Josie and Jer’s analyses show how the emulation platform helped to document specific interactions and to “cite them,” so to speak. As shown by the analyses and speculative exercises documented above, the students engaged in detailed and patient observation of their historical system, made comparisons across versions, and provided deeper, phenomenal perspectives of the systems themselves. These myriad analytical affordances portend emulation’s potential to yield a deeper understanding of historical software, and to foster insights relevant to the development of new systems.

In approaching the emulations, we observed that students followed their own analytical and creative inclinations. For example, while Jer’s analysis was conducted by systematically loading example files and teasing them apart, Josie’s consisted of creating designs entirely from scratch. This is possibly not just a reflection of their specific questions and intellectual styles, but also a factor that shaped their observations. During the two-week exercise, they spent approximately ten hours with their selected program inside the spartan EaaSI interface (See Figure 3 above). It’s worth noting that the EaaSI interactions did introduce a bit of latency and other mild impediments that would not have been present in historical context — for instance, at the time of the study there was no easy way to extract files from each environment — but the ability to experiment with the older interfaces proved compelling enough to overcome these frustrations.

It is also important to note that the results were likely shaped by each students’ level of proficiency with the systems they analyzed. As both Josie and Jer were expert users of modern versions of

AutoCAD, they were able to interpret the historical versions in a deeper way than if they were novices by, for example, identifying consistent elements across the different versions. In addition, their experience as practicing architects gave them tools to reflect meaningfully on the software's contexts of use. This is particularly clear in Jer's acute observations about how certain software features had concrete impacts on the organization of the architectural workplace. In contrast, analyses produced by students who were not expert users of their chosen systems seemed to elicit a different kind of reflection. These students developed an understanding of their chosen software's visual and interactive elements, but they were not as able to draw comparative insights, or to enrich their analyses with their personal experience, as Josie and Jer did so effectively in their readings. However, the exposure to a historical methodology based on software emulations was itself valuable. It offered them the opportunity to analyze a piece of software through interaction, and proved valuable as an element informing their speculative prototypes.

In the Background section we noted how historical engagements within CSCW can be seen to enact three broad trends — documentation, reflection, and critique. The work presented in this paper seems to more directly address the latter two categories. Although emulations can support the exhaustive documentation of a historical system's functionality — by rendering documentary file formats not compatible with modern systems — the analyses we presented did not aim at this type of work. Instead, they sought to deliver grounded reflections about some of AutoCAD's historic design choices, and a critique of those choices as they impacted AutoCAD's continuing development. In addition, they helped instigate ideas about the development of new systems.

6 CONCLUSION

Emulation, as we have shown here, can be part of a pedagogical and research toolkit for examining historical and sociotechnical aspects of software, and informing its conceptualization and design. Emulation affords access to visual, operational, and interactive aspects of software that other methods such as archival research, documentary analysis, and interviews cannot. Thus, emulation has the potential to enliven aspects of software research and learning with an expanded context that includes phenomenal engagements with interactive systems, their time-based operation, and their visual and operational qualities.

We traced these potentials in our analysis of the work of students in a university seminar exploring computer-aided design systems. Supported by a distributed emulation platform, one of the activities of the class involved the analyses of different computer-aided design systems. We focused on the analysis of different versions of AutoCAD by two class participants, which helped to trace the system's transition from an elemental drafting tool into a more sophisticated geometric modeling platform over the course of several decades, distilling important aspects of its evolution, core functionality, and interactivity. A contribution of this paper is thus to offer a detailed case study for the incorporation of emulations as a resource in software research, pedagogy, and design. An additional contribution is to situate emulation within a landscape of historical and interpretive methods within CSCW and adjacent fields. In contrast with other approaches to the study of historical and sociotechnical aspects of software — such as software reconstruction, trace ethnography, and the analysis of documentary and oral sources — emulation offers access to software systems as primary research materials, making their interface and operation available for hands-on analysis and experience.

As the analyses presented above show, software operations manifest not only as logical possibilities, but also as phenomenal experiences. Differences in how to save a document, or activate a particular command, illustrate how the use of an interface can be a deeply personal affair, intertwined with a user's disciplinary inclinations and personal preferences. These distinctions cannot be fully prescribed in processes of UX design which often rely on rigid conceptual models of user,

interface, and operation. Enriched by emulations, historical studies of software-based practices can examine not just the logical structures of the source code, but also the sensory and affective experiences of users. Emulation, our findings suggest, can help make these available for both critical scrutiny and creative intervention.

Of course, the experience of using these emulated interactive environments is not identical to that of using the legacy environments on their original hardware, and this constitutes a limitation of this method. Aside from the speed of executing instruction sets, hardware itself modulates the experience of interacting in a computing environment: the glow of a cathode ray tube monitor, the shape and weight of peripheral I/O devices, and the speed of network connectivity are important to the experience of computing, and to its history, but are not easily captured by emulation. Sensitive to this fact, some class participants mentioned that they would like more configuration options for the display and input devices, expressing frustration with mouse lag and input resolution. Replicating or maintaining obsolete hardware poses serious challenges for archivists and researchers alike [42] and is for many purposes impractical. While emulations cannot by themselves precisely reproduce the original tempo of a historical piece of software, nor the tactility of the hardware through which it was first experienced, they are viable routes for staging the “space of possible actions” afforded by historical software systems [25], and revealing them to both scholarly and design analysis.

We would like to suggest that the potential of emulation for software research and pedagogy is best realized in conjunction with other research methods and types of evidence. Technology researchers concerned with historical systems depend on forms of written, recorded, and tabulated evidence that are all necessarily partial. Dealing with this partiality of evidence is in fact a core concern of recent historiography [53]. By helping trace a software’s evolution and distinctive visual and interactive qualities, the analyses of different emulated legacy systems presented here offered a glimpse of how studies of computer-supported cooperative work might benefit from close readings of historical mainstream software interfaces and how these readings might enrich other forms of sociotechnical analysis and design.

ACKNOWLEDGMENTS

The authors wish to thank the students in the “Experimental Archaeologies of CAD” course at Carnegie Mellon University’s School of Architecture for their inquisitiveness and rigor, the Carnegie Mellon University Libraries for giving the class access to their EaaS node, and to the EaaS program at Yale University Library, funded by the Sloan Foundation and Mellon Foundation, for a small grant that offered crucial support for this research.

REFERENCES

- [1] [n.d.]. Dd. <https://pubs.opengroup.org/onlinepubs/9699919799/utilities/dd.html>.
- [2] [n.d.]. Guymager Homepage. <https://guymager.sourceforge.io/>.
- [3] Morgan G. Ames. 2019. *The Charisma Machine: The Life, Death, and Legacy of One Laptop per Child*. MIT Press.
- [4] Autodesk. 1984. AutoCAD Ad. *Scientific American* 251, 3 (1984), 225–226.
- [5] Thomas Bähr, Michelle Lindlar, Klaus Rechert, and Thomas Liebetaut. 2014. Functional Access to Electronic Media Collections Using Emulation-as-a-Service. In *Proceedings of the 11th International Conference on Digital Preservation, iPRES 2014, Melbourne, Australia, October 6 - 10, 2014*, Serena Coates, Ross King, Steve Knight, Christopher A. Lee, Peter McKinney, Erin O’Meara, and David Pearson (Eds.).
- [6] Fabrice Bellard. 2005. QEMU, A Fast and Portable Dynamic Translator. In *USENIX Annual Technical Conference, FREENIX Track*, Vol. 41. 41–46.
- [7] Susanne Bødker. 1993. Historical Analysis and Conflicting Perspectives — Contextualizing HCI. In *Human-Computer Interaction*, G. Goos, J. Hartmanis, Leonard J. Bass, Juri Gornostaev, and Claus Unger (Eds.). Vol. 753. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–10. https://doi.org/10.1007/3-540-57433-6_34
- [8] Yana Boeva, Devon Elliott, Edward Jones-Imhotep, Shezan Muhammedi, and William J. Turkel. 2017. Doing History by Reverse Engineering Electronic Devices. In *Making Things and Drawing Boundaries*, Jentery Sayers (Ed.). University of

- Minnesota Press, 163–176. <https://doi.org/10.5749/j.ctt1pwt6wq.21>
- [9] Geoffrey C Bowker. 1996. The History of Information Infrastructures: The Case of the International Classification of Diseases. *Information Processing & Management* 32, 1 (Jan. 1996), 49–61. [https://doi.org/10.1016/0306-4573\(95\)00049-M](https://doi.org/10.1016/0306-4573(95)00049-M)
- [10] Martin Campbell-Kelly. 1990. *The Airy Tape : An Early Chapter in the History of Debugging*. Report Number 153. University of Warwick. Department of Computer Science, Coventry, UK.
- [11] Daniel Cardoso Llach. 2015. *Builders of the Vision: Software and the Imagination of Design* (1 edition ed.). Routledge.
- [12] Daniel Cardoso Llach. 2019. Tracing Design Ecologies: Collecting and Visualizing Ephemeral Data as a Method in Design and Technology Studies. In *DigitalSTS: A Field Guide for Science & Technology Studies*. Princeton University Press, 451–471.
- [13] Daniel Cardoso Llach and Scott Donaldson. 2019. An Experimental Archaeology of CAD: Using Software Reconstruction to Explore the Past and Future of Computer-Aided Design. In *Computer-Aided Architectural Design. "Hello, Culture"*, Ji-Hyun Lee (Ed.). Springer, 105–119.
- [14] Gary Lee Downey. 2014. *The Machine in Me: An Anthropologist Sits Among Computer Engineers*. Routledge.
- [15] Thomas Elsaesser. 2016. *Film History as Media Archaeology: Tracking Digital Cinema*. Amsterdam University Press, Amsterdam.
- [16] Kristine K. Fallon. 1997. *The AEC Technology Survival Guide: Managing Today's Information Practice*. Wiley.
- [17] Chris Garcia. 2016. *Harold Cohen and AARON—A 40-Year Collaboration*. <https://computerhistory.org/blog/harold-cohen-and-aaron-a-40-year-collaboration/> Section: Curatorial Insights.
- [18] R. Stuart Geiger and David Ribes. 2011. Trace Ethnography: Following Coordination through Documentary Practices. In *2011 44th Hawaii International Conference on System Sciences*. 1–10. <https://doi.org/10.1109/HICSS.2011.455>
- [19] Lisa Gitelman. 2006. *Always Already New: Media, History, and the Data of Culture*. The MIT Press, Cambridge, MA.
- [20] Google. [n.d.]. *Cloud Speech-to-Text - Speech Recognition*. <https://cloud.google.com/speech-to-text> Library Catalog: cloud.google.com.
- [21] Stewart Granger. 2000. Digital Preservation & Emulation: From Theory to Practice. (2000), 8.
- [22] Stewart Granger. 2000. *Emulation as a Digital Preservation Strategy*. Technical Report. Corporation for National Research Initiatives.
- [23] Jonathan Grudin. 1990. THE COMPUTER REACHES OUT: THE HISTORICAL CONTINUITY OF INTERFACE DESIGN. (1990), 8.
- [24] Patrick Jagoda. 2014. Gaming the Humanities. *Differences* 25, 1 (2014), 189–215. <https://doi.org/doi.org/10.1215/10407391-2420045>.
- [25] Lars-Erik Janlert and Erik Stolterman. 2017. The Meaning of Interactivity: Some Proposals for Definitions and Measures. *Human Computer Interaction* 32 (2017), 103–138. <https://doi.org/10.1080/07370024.2016.1226139>
- [26] Anker Helms Jørgensen. 2007. Mainstream versus Mainstream: Two Approaches to User Interface History. In *Seventh Danish HCI Research Symposium*. 33–34.
- [27] Patricia Kain. 1998. *How to Do a Close Reading*. <https://writingcenter.fas.harvard.edu/pages/how-do-close-reading>
- [28] Eric Kaltman, Joseph Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2017. Getting the GISST: A Toolkit for the Creation, Analysis and Reference of Game Studies Resources. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. Hyannis, MA.
- [29] Vera Khovanskaya, Lynn Dombrowski, Jeffrey Rzeszotarski, and Phoebe Sengers. 2019. The Tools of Management: Adapting Historical Union Tactics to Platform-Mediated Labor. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–22. <https://doi.org/10.1145/3359310>
- [30] R. Krishnamurti and C. Giraud. 1986. Towards a Shape Editor: The Implementation of a Shape Generation System. *Environment and Planning B: Planning and Design* 13, 4 (1986), 391–404. <https://doi.org/10.1068/b130391>
- [31] Antoine Lamé. [n.d.]. *Gaze Tracking*. <https://github.com/antoinelame/GazeTracking> original-date: 2019-02-09T22:27:07Z.
- [32] Thomas Larsson and Ulf Stenlund. 2008. Vignettes of Sweden's Computer Graphics History. *IEEE Annals of the History of Computing* 30, 1 (Jan. 2008), 50–58. <https://doi.org/10.1109/MAHC.2008.16>
- [33] Christopher A. Lee, Matthew Kirschenbaum, Alexandra Chassanoff, Porter Olsen, and Kam Woods. 2012. Bitcurator: Tools and Techniques for Digital Forensics in Collecting Institutions. *D-Lib Magazine* 18, 5/6 (2012), 14–21.
- [34] Paul McJones. 2017. In Search of the Original Fortran Compiler. *IEEE Annals of the History of Computing* 39, 2 (2017), 81–88. <https://doi.org/10.1353/ahc.2017.0015>
- [35] Nick Montfort. 2000. Cybertext Killed the Hypertext Star | Electronic Book Review. <http://www.electronicbookreview.com/thread/electropoetics/cyberdebates>.
- [36] Brad A. Myers. 1998. A Brief History of Human-Computer Interaction Technology. *interactions* 5, 2 (1998), 44–54. <https://doi.org/10.1145/274430.274436>
- [37] Software Preservation Network. [n.d.]. *Emulation-as-a-Service Infrastructure*. <https://www.softwarepreservationnetwork.org/projects/emulation-as-a-service-infrastructure/>

- [38] Trevor Owens and Thomas Padilla. 2020. Digital Sources and Digital Archives: Historical Evidence in the Digital Age. *International Journal of Digital Humanities* (May 2020). <https://doi.org/10.1007/s42803-020-00028-7>
- [39] William Lyman Porter. 1969. *The development of DISCOURSE: a language for computer assisted city design*. <http://dspace.mit.edu/handle/1721.1/39037>
- [40] Klaus Rechert, Thomas Liebetraut, Oleg Stobbe, Nathalie Lubetzki, and Tobias Steinke. 2017. The RESTful EMiL: Integrating Emulation into Library Reading Rooms. *Alexandria* 27, 2 (Aug. 2017), 120–136. <https://doi.org/10.1177/0955749017725427>
- [41] David Ribes and Jessica Beth Polk. 2012. Historical Ontology and Infrastructure. In *Proceedings of the 2012 iConference on - iConference '12*. ACM Press, Toronto, Ontario, Canada, 254–262. <https://doi.org/10.1145/2132176.2132209>
- [42] David SH Rosenthal. 2015. Emulation & Virtualization as Preservation Strategies. (2015).
- [43] Daniela K. Rosner. 2018. *Critical Fabulations: Reworking the Methods and Margins of Design*. MIT Press.
- [44] Dr Seamus Ross. 2000. Changing Trains at Wigan: Digital Preservation and the Future of Scholarship. (2000), 47.
- [45] Jeff Rothenberg. 1995. Ensuring the Longevity of Digital Documents. *Scientific American* 272, 1 (Jan. 1995), 42–47. <https://doi.org/10.1038/scientificamerican0195-42>
- [46] James E. Smith. 2005. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann Publishers, Amsterdam ; Boston.
- [47] MacKenzie Smith. 2009. Curating Architectural 3D CAD Models. *International Journal of Digital Curation* 4, 1 (2009), 98–106. <https://doi.org/10.2218/ijdc.v4i1.81>
- [48] Robert Soden, David Ribes, Maggie Jack, Will Sutherland, Vera Khovanskaya, Seyram Ayle, Phoebe Sengers, and Susanne Bø dker. 2019. Fostering Historical Research in CSCW & HCI. In *Conference Companion Publication of the 2019 on Computer Supported Cooperative Work and Social Computing - CSCW '19*. ACM Press, Austin, TX, USA, 517–522. <https://doi.org/10.1145/3311957.3359436>
- [49] David Walden and Imp Software Guys. 2014. The Arpanet IMP Program: Retrospective and Resurrection. *IEEE Annals of the History of Computing* 36, 2 (April 2014), 28–39. <https://doi.org/10.1109/MAHC.2014.30>
- [50] John Walker. 1990. *AutoDESK File*. New Riders Publishing, USA.
- [51] David E. Weisberg. 2008. Autodesk and AutoCAD. In *The Engineering Design Revolution: The People, Companies and Computer Systems That Changed Forever the Practice of Engineering*. 1–51.
- [52] Hayden White. 1988. Historiography and Historiophoty. *The American Historical Review* 93, 5 (1988), 1193–1199.
- [53] Hayden White. 2013. History as Fulfillment. In *Philosophy of History After Hayden White*, Robert Doran (Ed.). Bloomsbury Academic, London.
- [54] Stephen Williams. 2013. *Improving Usability of Pedagogical Computer Emulation Interfaces*. Ph.D. Dissertation. <https://doi.org/10.13140/RG.2.1.2262.1202>
- [55] Susan Wyche, Phoebe Sengers, and Rebecca E. Grinter. 2006. Historical Analysis: Using the Past to Design the Future. In *UbiComp 2006: Ubiquitous Computing*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Paul Dourish, and Adrian Friday (Eds.). Vol. 4206. Springer Berlin Heidelberg, Berlin, Heidelberg, 35–51. https://doi.org/10.1007/11853565_3
- [56] Albena Yaneva. 2012. *Mapping Controversies in Architecture*. Ashgate Publishing, Ltd.

Received October 2020; revised April 2021; accepted May 2021