Manovich, Lev. "Introduction." *Software Takes Command*. New York: Bloomsbury, 2013. 1–52. *Bloomsbury Collections*. Web. 26 Apr. 2023. <<http://dx.doi.org/10.5040/9781472544988.0005>>.

Downloaded from Bloomsbury Collections, www.bloomsburycollections.com, 26 April 2023, 13:19 UTC.

Copyright © Lev Manovich 2013. You may share this work for non-commercial purposes only, provided you give attribution to the copyright holder and the publisher, and provide a link to the Creative Commons licence.

Introduction

Understanding media

I called my earlier book-length account of the new cultural forms enabled by computerization *The Language of New Media* (completed in 1999, it came out in 2001). By that time, the process of adoption of software-based tools in all areas of professional media production was almost complete, and “new media art” was in its heroic and vibrant stage—offering many possibilities not yet touched by commercial software and consumer electronics.

Ten years later, most media became “new media.” The developments of the 1990s have been disseminated to the hundreds of millions of people who are writing blogs, uploading photos and videos to media sharing sites, and use free media authoring and editing software tools that ten years earlier would have cost tens of thousands of dollars.

Thanks to the practices pioneered by Google, the world is now used to running on web applications and services that have never been officially completed but remain forever in Beta stage. Since these applications and services run on the remote servers, they can be updated anytime without consumers having to do anything—and in fact, Google is updating its search algorithm code a few times a day. Similarly, Facebook is also updating its code daily, and sometimes it breaks. (Facebook’s motto expressed in posters around its offices is “Move Fast and Break Things.”) Welcome to the world of permanent change—the world that is now defined

not by heavy industrial machines that change infrequently, but by software that is always in flux.

Why should humanists, social scientists, media scholars, and cultural critics care about software? Because outside of certain cultural areas such as crafts and fine art, software has replaced a diverse array of physical, mechanical, and electronic technologies used before the twenty-first century to create, store, distribute and access cultural artifacts. When you write a letter in Word (or its open source alternative), you are using software. When you are composing a blog post in Blogger or WordPress, you are using software. When you tweet, post messages on Facebook, search through billions of videos on YouTube, or read texts on Scribd, you are using software (specifically, its category referred to as “web applications” or “webware”—software which is accessed via web browsers and which resides on the servers).

And when you play a video game, explore an interactive installation in a museum, design a building, create special effects for a feature film, design a website, use a mobile phone to read a movie review or to view the actual movie, and carry out thousands of other cultural activities, in practical terms, you are doing the same thing—using software. Software has become our interface to the world, to others, to our memory and our imagination—a universal language through which the world speaks, and a universal engine on which the world runs. What electricity and the combustion engine were to the early twentieth century, software is to the early twenty-first century.

This book is concerned with “media software”—programs such as Word, PowerPoint, Photoshop, Illustrator, After Effects, Final Cut, Firefox, Blogger, WordPress, Google Earth, Maya, and 3ds Max. These programs enable creation, publishing, sharing, and remixing of images, moving image sequences, 3D designs, texts, maps, and interactive elements, as well as various combinations of these elements such as websites, interactive applications, motion graphics, virtual globes, and so on. Media software also includes web browsers such as Firefox and Chrome, email and chat programs, news readers, and other types of software applications whose primary focus is accessing media content (although they sometimes also include some authoring and editing features.)

These software tools for creating, interacting with, and sharing media represent a particular subset of application software

A digital studio in Seoul, South Korea, 1/2006. This small studio was responsible for the photography of all Samsung phones, to appear in its ads worldwide. In the photos we see studio staff adjusting the phone photos in Photoshop. Later these high-resolution retouched images were inserted in the Samsung TV ad, thus assuring that the product details are clearly visible.

(including web applications) in general. Given this, we may expect that all these tools inherit certain “traits” common to all contemporary software. Does this mean that regardless of whether you are working on designing a space, creating special effects for a feature film, designing a website, or making information graphics, your design process may follow a similar logic? Are there some structural features which motion graphics, graphic designs, websites, product designs, buildings, and video games share since they are all designed with software? More generally, how are interfaces and the tools of media authoring software shaping the contemporary aesthetics and visual languages of different media forms?

Behind these questions investigated in this book lies another theoretical question. This question drives the book narrative and motivates my choice of topics. What happens to the idea of a “medium” after previously media-specific tools have been simulated and extended in software? Is it still meaningful to talk about different mediums at all? Or do we now find ourselves in a new brave world of one single monomedium, or a metamedium (to borrow the term of the book’s key protagonist Alan Kay)?

In short: *What is “media” after software?*

Does “media” still exist?

This book is a theoretical account of media software and its effects on the practice and the very concept of media. Over the last two decades, software has replaced most other media technologies that emerged in the nineteenth and twentieth centuries. Today it is ubiquitous and taken for granted—and yet, surprisingly, few people know about its history and the theoretical ideas behind its development. You are likely to know the names of Renaissance artists who popularized the use of linear perspective in western art (Brunelleschi, Alberti) or early twentieth-century inventors of modern film language (D. W. Griffith, Eisenstein, etc.)—but I bet you do not know where Photoshop comes from, or Word, or any other media tool you are using every day. More importantly, you probably do not know why these tools were invented in the first place.

What is the intellectual history of media software? What was the thinking and motivation of the key people and research groups they were directing—J. C. R. Licklider, Ivan Sutherland, Ted

Nelson, Douglas Engelbart, Alan Kay, Nicholas Negroponte—who between 1960 and the late 1970s created most of the concepts and practical techniques that underlie today’s media applications? As I discovered—and I hope you will share my original surprise, in reading my analysis of the original texts by these people—they were as much media theoreticians as computer engineers. I will discuss their media theories and test them in view of the digital media developments in the subsequent decades. As we will see, the theoretical ideas of these people and their collaborators work very well today, helping us to better understand the contemporary software we use to create, read, view, remix, and share.

Welcome, then, to the “secret history” of our software culture—secret not because it was deliberately hidden but because until recently, excited by all the rapid transformations cultural computerization was bringing about, we did not bother to examine its origins. This book will try to convince you that such an examination is very much worth your time.

Its title pays homage to a seminal twentieth-century book *Mechanization Takes Command: a Contribution to Anonymous History* (1947) by architectural historian and critic Sigfried Giedion. In this work Giedion traces the development of mechanization in industrial society across a number of domains, including systems of hygiene and waste management, fashion, agricultural production, and food system, with separate sections of the book devoted to bread, meat, and refrigeration. Much more modest in scope, my book presents episodes from the history of “softwarization” (my neologism) of culture between 1960 and 2010, with a particular attention to media software—from the original ideas which led to its development to its current ubiquity.

My investigation is situated within a broader intellectual paradigm of “software studies.” From this perspective, this book’s contribution is the analysis of the ideas that eventually led to media software, and the effects of the adoption of this type of software on contemporary media design and visual culture.

Note that the category *media software* is a subset of the category *application software*; this category in its turn is a subset of the category *software*¹—which I understand to include

¹ http://en.wikipedia.org/wiki/List_of_software_categories (July 7, 2011).

not only application software, system software, and computer programming tools, but also social network services and social media technologies.²

If we understand software in this extended sense, we can ask, What does it mean to live in “software society”? And what does it mean to be part of “software culture”? These are the questions the next section will take up.

Software, or the engine of contemporary societies

In the beginning of the 1990s, the most famous global brands were the companies that were in the business of producing materials or goods, or processing physical matter. Today, however, the lists of best-recognized global brands are topped with the names such as Google, Facebook, and Microsoft. (In fact, in 2007 Google became number one in the world in terms of brand recognition.) And, at least in the US, the most widely read newspapers and magazines—*New York Times*, *USA Today*, *Business Week*, etc.—feature daily news and stories about Facebook, Twitter, Apple, Google, and other IT companies.

What about other media? When I was working on the first draft of this book in 2008, I checked the business section of the CNN website. Its landing page displayed market data for just ten companies and indexes.³ Although the list was changed daily, it was always likely to include some of the same IT brands. Let us take January 21, 2008 as an example. On that day the CNN list contained the following companies and indexes: Google, Apple, S&P 500 Index, Nasdaq Composite Index, Dow Jones Industrial

² Andreas Kaplan and Michael Haenlein define social media as “a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0, which allows the creation and exchange of user-generated content.” Andreas Kaplan and Michael Haenlein, “Users of the world, unite! The challenges and opportunities of Social Media,” *Business Horizons* 53, no. 1 (January–February 2010), pp. 59–68, <http://dx.doi.org/10.1016/j.bushor.2009.09.003>

³ <http://money.cnn.com> (January 21, 2008).

Average, Cisco Systems, General Electric, General Motors, Ford, Intel.⁴

This list is very telling. The companies that deal with physical goods and energy appear in the second part of the list: General Electric, General Motors, Ford. Right before and after these three, we see two IT companies that provide hardware: Intel makes computer chips, while Cisco makes network equipment. What about the two companies which are on top: Google and Apple? The first is in the business of information (“Google’s mission is to organize the world’s information and make it universally accessible and useful”⁵), while the second is making consumer electronics: phones, tablets, laptops, monitors, music players, etc. But actually, they are both making something else. And apparently, this something else is so crucial to the workings of US economy—and consequently, global world as well—that these companies almost daily appear in business news. And the major Internet companies that also appear daily in news such as Google, Facebook, Twitter, Amazon, eBay, and Yahoo, are in the same business.

This “something else” is *software*. Search engines, recommendation systems, mapping applications, blog tools, auction tools, instant messaging clients, and, of course, platforms which allow people to write new software—iOS, Android, Facebook, Windows, Linux—are in the center of the global economy, culture, social life, and, increasingly, politics. And this “cultural software”—cultural in a sense that it is directly used by hundreds of millions of people and that it carries “atoms” of culture—is only the visible part of a much larger software universe.

In *Software Society* (2003), an unrealized book proposal put together by me and Benjamin Bratton, we described the importance of software and its relative invisibility in humanities and social science research:

Software controls the flight of a smart missile toward its target during war, adjusting its course throughout the flight. Software runs the warehouses and production lines of Amazon, Gap, Dell, and numerous other companies allowing them to assemble and dispatch material objects around the world, almost in no

⁴ *Ibid.*

⁵ <http://www.google.com/about/company/> (September 23, 2012).

time. Software allows shops and supermarkets to automatically restock their shelves, as well as automatically determine which items should go on sale, for how much, and when and where in the store. Software, of course, is what organizes the Internet, routing email messages, delivering Web pages from a server, switching network traffic, assigning IP addresses, and rendering Web pages in a browser. The school and the hospital, the military base and the scientific laboratory, the airport and the city—all social, economic, and cultural systems of modern society—run on software. Software is the invisible glue that ties it all together. While various systems of modern society speak in different languages and have different goals, they all share the syntaxes of software: control statements “if then” and “while do,” operators and data types (such as characters and floating point numbers), data structures such as lists, and interface conventions encompassing menus and dialog boxes.

If electricity and the combustion engine made industrial society possible, software similarly enables global information society. The “knowledge workers,” the “symbol analysts,” the “creative industries,” and the “service industries”—none of these key economic players of the information society can exist without software.

Examples are data visualization software used by a scientist, spreadsheet software used by a financial analyst, Web design software used by a designer working for a transnational advertising agency, or reservation software used by an airline. Software is what also drives the process of globalization, allowing companies to distribute management nodes, production facilities, and storage and consumption outputs around the world. Regardless of which new dimension of contemporary existence a particular social theory of the last few decades has focused on—information society, knowledge society, or network society—all these new dimensions are enabled by software.

Paradoxically, while social scientists, philosophers, cultural critics, and media and new media theorists seem by now to cover all aspects of IT revolution, creating a number of new disciplines such as cyberculture studies, Internet studies, game studies, new media theory, digital culture, and digital humanities, the underlying engine which drives most of these subjects—software—has received comparatively little attention.

Even today, ten years later, when people are constantly interacting with and updating dozens of apps on their mobile phones and other computer devices, *software as a theoretical category* is still invisible to most academics, artists, and cultural professionals interested in IT and its cultural and social effects.

There are some important exceptions. One is the open source movement and related issues around copyright and IP that have been extensively discussed in many academic disciplines. We also see a steadily growing number of trade books about Google, Facebook, Amazon, eBay, Oracle, and other web giants. Some of these books offer insightful discussions of the software developed by these companies and the social, political, cognitive, and epistemological effects of this software. (For a good example, see John Battelle, *The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture*.⁶)

So while we are in a better situation today when we put together our proposal for *Software Society* in 2003, I feel that it is still meaningful to quote it (the only additions are the references to “social media” and “crowdsourcing”):

If we limit critical discussions of digital culture to the notions of “open access,” “peer production,” “cyber,” “digital,” “Internet,” “networks,” “new media,” or “social media,” we will never be able to get to what is behind new representational and communication media and to understand what it really is and what it does. If we don’t address software itself, we are in danger of always dealing only with its effects rather than the causes: the output that appears on a computer screen rather than the programs and social cultures that produce these outputs. “Information society,” “knowledge society,” “network society,” “social media,” “online collaboration,” “crowdsourcing”—regardless of which new feature of contemporary existence a particular analysis has focused on, all these new features are enabled by software. It is time we focused on software itself.

A similar sentiment is expressed in Noah Wardrip-Fruin’s *Expressive Processing* (2009) when he says in relation to books

⁶ John Battelle, *The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture* (Portfolio Trade, 2006).

about digital literature: “almost all of these have focused on what the machines of digital media look like from the outside: their output... regardless of perspective, writings on digital media almost all ignore something crucial: the actual processes that make digital media work, the computational machines that make digital media possible.”⁷ My book discusses what I take to be the key part of these “machines” today (because it is the only part which most users see and use directly): application software.

What is software studies?

This book aims to contribute to the developing intellectual paradigm of “software studies.” What is software studies? Here are a few definitions. The first comes from my *The Language of New Media*, where, as far as I know, the terms “software studies” and “software theory” appeared for the first time. I wrote, “New media calls for a new stage in media theory whose beginnings can be traced back to the revolutionary works of Robert Innis and Marshall McLuhan of the 1950s. To understand the logic of new media we need to turn to computer science. It is there that we may expect to find the new terms, categories, and operations that characterize media that became programmable. From media studies, we move to something which can be called software studies; from media theory—to software theory.”

Reading this statement today, I feel some adjustments are in order. It positions computer science as a kind of absolute truth, a given which can explain to us how culture works in software society. But computer science is itself part of culture. Therefore, I think that Software Studies has to investigate the role of software in contemporary culture, and the cultural and social forces that are shaping the development of software itself.

The book that first comprehensively demonstrated the necessity of the second approach was *New Media Reader* edited by Noah Wardrip-Fruin and Nick Montfort (The MIT Press, 2003). The publication of this groundbreaking anthology laid the framework

⁷ Noah Wardrip-Fruin, *Expressive Processing* (Cambridge, MA: The MIT Press, 2009).

for the historical study of software as it relates to the history of culture. Although *Reader* did not explicitly use the term “software studies,” it did propose a new model for how to think about software. By systematically juxtaposing important texts by pioneers of cultural computing and artists and writers active in the same historical periods, *New Media Reader* demonstrated that both belonged to the same larger epistemes. That is, often the same idea was simultaneously articulated independently by artists and scientists who were inventing cultural computing. For instance, the anthology opens with a story by Jorge Borges (1941) and an article by Vannevar Bush (1945) which both contain the idea of a massive branching structure as a better way to organize data and to capture human experience.

In February 2006 Matthew Fuller who had already published a pioneering book on software as culture (*Behind the Blip: essays on the culture of software*, 2003) organized the very first *Software Studies Workshop* at Piet Zwart Institute in Rotterdam. Introducing the workshop, Fuller wrote, “Software is often a blind spot in the theorization and study of computational and networked digital media. It is the very grounds and ‘stuff’ of media design. In a sense, all intellectual work is now ‘software study’, in that software provides its media and its context, but there are very few places where the specific nature, the materiality, of software is studied except as a matter of engineering.”⁸

I completely agree with Fuller that, “all intellectual work is now ‘software study.’” Yet it will take some time before the intellectuals will realize it. To help bring this change, in 2008, Matthew Fuller, Noah Wardrip-Fruin and I established the *Software Studies* book series at MIT Press. The already published books in the series are *Software Studies: A Lexicon* edited by Fuller (2008), *Expressive Processing: Digital Fictions, Computer Games, and Software Studies* by Wardrip-Fruin (2009), *Programmed Visions: Software and Memory* by Wendy Hui Kyong Chun (2011), *Code/Space: Software and Everyday Life* by Rob Kitchin and Martin Dodge (2011), and *Speaking Code: Coding as Aesthetic and Political Expression* by Geoff Cox and Alex Mclean (2012). In 2011, Fuller together with a number of UK researchers established

⁸ <http://pzwart.wdka.hro.nl/mdr/Seminars2/softstudworkshop> (January 21, 2008).

Computational Culture, an open-access peer-reviewed journal that provides a platform for more publications and discussions.

In addition to this series, I am also happy to see a growing number of other titles written from the perspectives of platform studies, digital humanities, cyberculture, internet studies, and game studies. Many of these books contain important insights and discussions which help us better understand the roles of software. Rather than trying to list all of them, I will only provide a few examples of works which exemplify the first two of these perspectives (more will be in press by the time you are reading this). Platform studies: Nick Montfort and Ian Bogost's *Racing the Beam: The Atari Video Computer System* (2009), Jimmy Maher's *The Future Was Here: The Commodore Amiga* (2012). Digital Humanities: *Mechanisms: New Media and the Forensic Imagination* (Matthew G. Kirschenbaum, 2008), *The Philosophy of Software: Code and Mediation in the Digital Age* (David Berry, 2011), *Reading Machines: Toward an Algorithmic Criticism* (Stephen Ramsay, 2011), *How We Think: Digital Media and Contemporary Technogenesis* (Katherine Hayles, 2012).⁹ Also highly relevant is the first book in what may become a new area of "format studies": *MP3: The Meaning of a Format* (Jonathan Sterne, 2012).¹⁰

Another set of works which are relevant to understanding the roles and functioning of software systems comes from people who were trained in computer science but are also equally at home in cultural theory, philosophy, digital art, or other humanistic fields: Phoebe Sengers, Warren Sack, Fox Harrell, Michael Mateas, Paul Dourish, and Phil Agre.

Yet another relevant category of books comprises the historical studies of important labs and research groups central to the development of modern software, other key parts of information technology such as the internet, and professional practices of

⁹ Nick Montfort and Ian Bogost, *Racing the Beam: The Atari Video Computer System* (The MIT Press, 2009); Jimmy Maher, *The Future Was Here: The Commodore Amiga* (The MIT Press, 2012); David Berry, *The Philosophy of Software: Code and Mediation in the Digital Age* (Palgrave Macmillan, 2011); Stephen Ramsay, *Reading Machines: Toward an Algorithmic Criticism* (University of Illinois Press, 2011), Katherine Hayles, *How We Think: Digital Media and Contemporary Technogenesis* (University of Chicago Press, 2012).

¹⁰ Jonathan Sterne, *MP3: The Meaning of a Format* (Duke University Press, 2012).

software engineering such as user testing. The examples of these works listed chronologically are Katie Hafner and Matthew Lyon's *Where Wizards Stay Up Late: The Origins Of The Internet* (1998), Michael Hiltzik's *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age* (2000), Martin Campbell-Kelly's *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (2004), and Nathan Ensmenger's *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (2010).¹¹

My all-time favorite book, however, remains *Tools for Thought* published by Howard Rheingold in 1985, right at the moment when domestication of computers and software starts, eventually leading to their current ubiquity. This book is organized around the key insight that computers and software are not just “technology” but rather the new *medium* in which we can think and imagine differently. Similar understanding was shared by all the heroes of this book who, with their collaborators, invented the computational “tools for thoughts”—J. C. R. Licklider, Ted Nelson, Douglass Engelbart, Bob Taylor, Alan Kay, Nicholas Negroponte. (Today many academics in humanities and social sciences still need to grasp this simple but fundamental idea. They continue to think of software as being strictly the domain of the Academic Computing Department in their universities—something which is only there to help them become more efficient, as opposed to the medium where human intellectual creativity now dwells.)

This short sketch of the intellectual landscape around software studies will be very incomplete if I do not mention the role of artists in pioneering the cultural discussions of software. Beginning around 2000, a number of artists and writers started to develop the practice of software art which included exhibitions, festivals, publishing books, and organizing online repositories of relevant works. The key figures in these developments were Amy Alexander,

¹¹ Katie Hafner and Matthew Lyon, *Where Wizards Stay Up Late: The Origins Of The Internet* (Simon & Schuster, 1998); Michael A. Hiltzik, *Dealers of Lightning: Xerox PARC and the Dawn of the Computer Age* (HarperBusiness, 2000); Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (The MIT Press, 2004); Nathan L. Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (The MIT Press, 2010).

Inke Arns, Adrian Ward, Geoff Cox, Florian Cramer, Matthew Fuller, Olga Goriunova, Alex McLean, Alessandro Ludovico, Pit Schultz, and Alexei Shulgin. In 2002 Christiane Paul organized CODEDOC—an exhibition of artistic code—at The Whitney Museum of American Art;¹² in 2003, the major festival of digital art Ars Electronica chose “Code” as its topic; and since 2001, the transmediale festival has included “artistic software” as one of its categories, and devoted a significant space to it in the festival’s symposiums. Some of the software art projects pioneered the examination of code as the new cultural and social artifact; others offered critical commentary on commercial software practices. For example, Adrian Ward created an ironic Auto-Illustrator—“an experimental, semi-autonomous, generative software artwork and a fully functional vector graphic design application to sit alongside your existing professional graphic design utilities.”

Recognizing that the bits of software studies exist across many books and art projects, Fuller writes in the Foreword to The MIT Press Software Studies book series:

Software is deeply woven into contemporary life—economically, culturally, creatively, politically—in manners both obvious and nearly invisible. Yet while much is written about how software is used, and the activities that it supports and shapes, thinking about software itself has remained largely technical for much of its history. Increasingly, however, artists, scientists, engineers, hackers, designers, and scholars in the humanities and social sciences are finding that for the questions they face, and the things they need to build, an expanded understanding of software is necessary. For such understanding they can call upon a strand of texts in the history of computing and new media, they can take part in the rich implicit culture of software, and they also can take part in the development of an emerging, fundamentally transdisciplinary, computational literacy. These provide the foundation for Software Studies.¹³

Indeed, a number of earlier works by the leading media theorists of our times—Friedrich A. Kittler, Peter Weibel, Katherine Hayles,

¹² <http://artport.whitney.org/commissions/codedoc/>

¹³ Matthew Fuller, Software Studies series introduction, <http://mitpress.mit.edu/catalog/browse/browse.asp?btype=6&csid=179> (July 14, 2011).

Lawrence Lessig, Manuel Castells, Alex Galloway, and others—can also be retroactively identified as belonging to software studies.¹⁴ Therefore, I strongly believe that this paradigm has already existed for a number of years but it has not been explicitly named until a few years ago.

In his introduction to a 2006 Rotterdam workshop Fuller pointed out that “software can be seen as an object of study and an area of practice for art and design theory and the humanities, for cultural studies and science and technology studies and for an emerging reflexive strand of computer science.” Since a new academic discipline can be defined either through a unique object of study, a new research method, or a combination of the two, how shall we think of software studies? Fuller’s statement implies that “software” is a new object of study which should be put on the agenda of existing disciplines and which can be studied using already existing methods—for instance, actor-network theory, social semiotics, or media archaeology.

There are good reasons for supporting this perspective. I think of software as *a layer that permeates all areas of contemporary societies*. Therefore, if we want to understand contemporary techniques of *control, communication, representation, simulation, analysis, decision-making, memory, vision, writing, and interaction*, our analysis cannot be complete until we consider this software layer. Which means that all disciplines which deal with contemporary society and culture—architecture, design, art criticism, sociology, political science, art history, media studies, science and technology studies, and all others—need to account for the role of software and its effects in whatever subjects they investigate.

At the same time, the existing work in software studies already demonstrates that if we are to focus on software itself, we need new methodologies. That is, it helps to practice what one writes about. It is not accidental that all the intellectuals who have most systematically written about software’s roles in society and culture have either programmed themselves or have been involved in cultural projects and practices which include writing and teaching software—for instance, Ian Bogost, Jay Bolter, Florian Cramer, Wendy Chun, Matthew Fuller, Alexander Galloway, Katherine

¹⁴ See Michael Truscello, review of “Behind the Blip: Essays on the Culture of Software,” *Cultural Critique* 63, Spring 2006, pp. 182–7.

Hayles, Matthew Kirschenbaum, Geert Lovink, Peter Lunenfeld, Adrian Mackenzie, Paul D. Miller, William J. Mitchell, Nick Montfort, Janet Murray, Katie Salen, Bruce Sterling, Noah Wardrip-Fruin, and Eric Zimmerman. In contrast, the scholars without this technical experience or involvement—for example, Manuel Castells, Bruno Latour, Paul Virilio, and Siegfried Zielinski—have not included discussions of software in their otherwise theoretically precise and highly influential accounts of modern media and technology.

In the 2000s, the number of students in media art, design, architecture, and humanities who use programming or scripting in their work has grown substantially—at least in comparison with 1999 when I first mentioned “software studies” in *The Language of New Media*. Outside of culture and academic industries, many more people today are also writing software. To a significant extent, this is the result of new programming and scripting languages such as ActionScript, PHP, Perl, Python, and Processing. Another important factor is the publication of APIs by all major Web 2.0 companies in the middle of the 2000s. (API, or Application Programming Interface, is a code that allows other computer programs to access services offered by an application. For instance, people can use Google Maps API to embed full Google Maps on their own websites.) These programming and scripting languages and APIs did not necessarily make programming easier. Rather, they made it much more efficient. For instance, since a young designer can create an interesting work with only couple of dozen lines of code written in Processing versus writing a really long Java program, s/he is much more likely to take up programming. Similarly, if only a few lines in JavaScript allows you to integrate all the functionality offered by Google Maps into your site, this is a great motivation for beginning to work with JavaScript. Yet another reason for more people writing software today is the emergence of a massive mobile apps marketplace that, unlike the desktop market, is not dominated by a few large companies. According to informal reports in the beginning of 2012, one million programmers were creating apps for the iOS platform (iPad and iPhone) alone, and another one million were doing this for the Android platform.

In his 2006 article covering new technologies that allow people with very little or no programming experience to create new custom software (such as Ning), Martin LaMonica wrote about a

future possibility of “a long tail for apps.”¹⁵ A few years later, this is exactly what happened. In September 2012, 700,000 apps were available on Apple App Store,¹⁶ and over 600,000 Android apps on Google Play.¹⁷

In the article called “A Surge in Learning the Language of the Internet” (March 27, 2012), the *New York Times* reported that, “The market for night classes and online instruction in programming and Web construction, as well as for iPhone apps that teach, is booming.” The article quoted Zach Sims, one of the founders of Codecademy (a web school which teaches programming through interactive lessons) who explained one of the reasons for this growing interest in learning programming and web design: “People have a genuine desire to understand the world we now live in. They do not just want to use the Web; they want to understand how it works.”¹⁸

In spite of these impressive developments, the gap between people who can program and who cannot remains—as does the gap between professional programmers and people who just took one or two short programming classes. Clearly, today the consumer technologies for capturing and editing media are much easier to use than even the most friendly programming and scripting languages. But it does not necessarily have to stay this way. Think, for instance, of what it took to set up a photo studio and take photographs in the 1850s versus simply pressing a single button on a digital camera or a mobile phone in the 2000s. Clearly, we are very far from such simplicity in programming. But I do not see any logical reasons why programming cannot one day become equally easy.

For now, the number of people who can script and program keeps increasing. Although we are far from a true “long tail” for software, software development is gradually getting more democratized. It is, therefore, the right moment to start thinking

¹⁵ Martin LaMonica, “The do-it-yourself Web emerges,” *CNET News*, July 31, 2006, http://www.news.com/The-do-it-yourself-Web-emerges/2100-1032_3-6099965.html

¹⁶ <http://www.mobilestatistics.com/mobile-statistics> (July 30, 2012).

¹⁷ <http://play.google.com/about/apps/> (July 30, 2012).

¹⁸ Jenna Wortham, “A Surge in Learning the Language of the Internet,” *New York Times*, March 27, 2012, <http://www.nytimes.com/2012/03/28/technology/for-an-edge-on-the-internet-computer-code-gains-a-following.html>

```
void setup()
{
  size(600,600);
  smooth();
  noLoop();
}

void draw()
{
  background(255);
  strokeWeight(10);
  translate(width/2,height-20);
  branch(0);
}

void branch(int depth){
  if (depth < 12) {
    line(0,0,0,-height/10);
    {
      translate(0,-height/10);
      rotate(random(-0.1,0.1));

      if (random(1.0) < 0.6){ // branching
        rotate(0.3);
        scale(0.7);
        pushMatrix();
        branch(depth + 1);
        popMatrix();
        rotate(-0.6);
        pushMatrix();
        branch(depth + 1);
        popMatrix();
      }
      else { // continue
        branch(depth);
      }
    }
  }
}
```

The complete code for tree_recursion, a Processing sketch by Mitchell Whitelaw, 2011, <http://www.openprocessing.org/sketch/8752>

Tree variations generated by tree_recursion code.

theoretically about how software is shaping our culture, and how it is shaped by culture in its turn. The time for “software studies” has arrived.

Cultural software

German media and literary theorist Friedrich Kittler wrote that students today should know at least two software languages: only “then they’ll be able to say something about what ‘culture’ is at the moment.”¹⁹ Kittler himself programmed in an assembler language—which probably determined his distrust of Graphical User Interfaces and modern software applications that use these interfaces. In a classical modernist move, Kittler argued that we need to focus on the “essence” of the computer—which for Kittler meant its mathematical and logical foundations and its early history characterized by tools such as assembler languages.

This book is determined by my own history of engagement with computers as a programmer, computer animator and designer, media artist, and as a teacher. This involvement started in the early 1980s, which was the decade of procedural programming (Pascal), rather than assembly programming. It was also the decade that saw the introduction of PCs, the emergence and popularization of desktop publishing, and the use of hypertext by some literary scholars. In fact, I came to NYC from Moscow in 1981, which was the year IBM introduced their first PC. My first experience with computer graphics was in 1983–4 on Apple IIe. In 1984 I saw a Graphical User Interface in its first successful commercial implementation on an Apple Macintosh. The same year I got a job at Digital Effects, one of the first computer animation companies in the world, where I learned how to program 3D computer models and animations. In 1986 I was writing computer programs that automatically processed photographs to make them look like paintings. In January 1987 Adobe Systems shipped Illustrator,

¹⁹ Friedrich Kittler, ‘Technologies of Writing/Rewriting Technology,’ *Auseinander* 1, no. 3 (Berlin, 1995), quoted in Michael Truscello, “The Birth of Software Studies: Lev Manovich and Digital Materialism,” *Film-Philosophy* 7, no. 55 (December 2003), <http://www.film-philosophy.com/vol7-2003/n55truscello.html>

followed by Photoshop in 1989. The same year saw the release of *The Abyss*, directed by James Cameron. This movie used pioneering CGI to create the first complex virtual character. And, by Christmas of 1990, Tim Berners-Lee had already created all the components of the World Wide Web as it exists today: a web server, web pages, and a web browser.

In short, during one decade the computer moved from being a culturally invisible technology to being the new engine of culture. While the progress in hardware and Moore's Law played crucial roles in this development, even more crucial was the release of software with a Graphical User Interface (GUI) aimed at non-technical users, word processing, applications for drawing, painting, 3D modeling, animation, music composing and editing, information management, hypermedia and multimedia authoring (HyperCard, Director), and global networking (World Wide Web). With easy-to-use software in place, the stage was set for the next decade of the 1990s when most culture industries—graphic design, architecture, product design, space design, filmmaking, animation, media design, music, higher education, and culture management—gradually adapted software tools. Thus, although I first learned to program in 1975 when I was in high school in Moscow, my take on software studies has been shaped by watching how during the 1980s GUI-based software quickly put the computer in the center of culture.

If software is indeed the contemporary equivalent of the combustion engine and electricity in terms of its social effects, every type of software needs to be taken into account. We need to consider not only "visible" software used by consumers but also "grey" software, which runs all systems and processes in contemporary society. However, since I do not have personal experience writing logistics software, industrial automation software, and other "grey" software, I will be not be writing about such topics. My concern is with a particular subset of software which I used and taught in my professional life. I call it *cultural software*.

While the term "cultural software" was previously used metaphorically (see J. M. Balkin, *Cultural Software: A Theory of Ideology*, 2003), I am going to use it literally to refer to certain types of software that support actions we normally associate with "culture." These cultural actions enabled by software can be divided into a number of categories (of course we should keep in

Adobe Photoshop, Macintosh version 1.0.7, 1990. Top: preferences window. Bottom: workspace.

mind that this is just one possible specific categorization system among many).

- 1 *Creating cultural artifacts and interactive services which contain representations, ideas, beliefs, and aesthetic values* (for instance, editing a music video, designing a package for a product, designing a website or an app).
- 2 *Accessing, appending, sharing, and remixing such artifacts (or their parts) online* (for instance, reading newspaper on the web, watching YouTube video, adding comments to a blog post).
- 3 *Creating and sharing information and knowledge online* (for instance, editing a Wikipedia article, adding places in Google Earth, including a link in a tweet).
- 4 *Communicating with other people* using email, instant message, voice-over IP, online text and video chat, social networking features such as wall postings, pokes, events, photo tags, notes, places, etc.
- 5 *Engaging in interactive cultural experiences* (for instance, playing a computer game).
- 6 *Participating in the online information ecology by expressing preferences and adding metadata* (for instance, automatically generating new information for Google Search whenever you use this service; clicking the “+1” button on Google+ or the “Like” button on Facebook; using the “retweet” function on Twitter).
- 7 *Developing software tools and services that support all these activities* (for instance, programming a library for Processing that enables sending and receiving data over the Internet;²⁰ writing a new plugin for Photoshop, creating a new theme for WordPress).

Technically, this software may be implemented in a variety of ways. Popular implementations (referred to in the computer industry as “software architecture”) include stand-alone applications that run on the user’s computing device, distributed applications (a client

²⁰ <http://www.processing.org/reference/libraries/> (July 7, 2011).

running on the user's device communicates with software on the server), and peer-to-peer networks (each computer becomes both a client and a server). If all this sounds completely unfamiliar, do not worry: all you need to understand is that "cultural software" as I will use this term covers a wide range of products and network services, as opposed to only single desktop applications such as Illustrator, Photoshop or After Effects that dominated media authoring in the 1990s and 2000s. For example, social network services such as Facebook and Twitter include multiple programs and databases running on company servers (for instance, in 2007 Google was running over one million servers around the world according to one estimate²¹) and the programs (called "clients") used by people to send emails, chat, post updates, upload video, leave comments, and perform other tasks on these services. (For instance, one can access Twitter using twitter.com, or tweetdeck.com, Twitter apps for iOS, Android, and dozens of third party websites and apps.)

Media applications

Let us go through the software categories that support the first four types of cultural activities listed above in more detail.

The first category is *software for creating, editing, and organizing media content*. The examples are Microsoft Word, PowerPoint, Photoshop, Illustrator, InDesign, Final Cut, After Effects, Maya, Blender, Dreamweaver, Aperture, and other applications. This category is in the center of this book. The industry uses a number of terms to refer to this category such as "media authoring," "media editing," and "media development" but I am going to refer to this category by using a single summary term. I will simply call it *media software*.

The second category is *software for distributing, accessing, and combining (or "publishing," "sharing," and "remixing") media content on the web*. Think Firefox, Chrome, Blogger, WordPress, Tumblr, Pinterest, Gmail, Google Maps, YouTube, Vimeo and

²¹ Pandia Search & Social, "Google: one million servers and counting," <http://www.pandia.com/sew/481-gartner.html>

other web applications and services. Obviously, the first and second categories overlap—for example, many desktop media applications allow you to upload your creations directly to popular media sharing sites, while many web applications and services include some authoring and editing functions (for example, YouTube has a built-in video editor). And blogging platforms and email clients sit right in the middle—they are used as much for publishing as for creating new content.

I will take for granted that since we all use application programs, or “apps,” we have a basic understanding of this term. Similarly, I also assume that we understand what “content” refers to in digital culture, but just to be sure, here are a couple of ways to define it. We can simply list various types of media which are created, shared, and accessed with media software and the tools provided by social media and sites: texts, images, digital video, animations, 3D objects and scenes, maps, as well as various combinations of these and other media. Alternatively, we can define “content” by listing genres, for instance, web pages, tweets, Facebook updates, casual games, multiplayer online games, user-generated video, search engine results, URLs, map locations, shared bookmarks, etc.

Digital culture tends to modularize content, i.e., enabling users to create, distribute, and re-use discrete content elements—looping animations to be used as backgrounds for videos, 3D objects to be used in creating complex 3D animations, pieces of code to be used in websites and blogs, etc.¹⁶ (This modularity parallels the fundamental principle of modern software engineering to design computer programs from small reusable parts called functions or procedures.) All such parts also qualify as “content.”

Between the late 1970s and the middle of the 2000s, application programs for media editing were designed to run on a user’s computer (minicomputers, PCs, scientific workstations, and later, laptops). In the next five years, companies gradually created more and more capable versions of these programs running in the “cloud.” Some of these programs are available via their own websites (Google Docs, Microsoft Web Office), while others are integrated with media hosting or social media services (e.g., Photobucket image and video editor). Many applications are implemented as clients that run on mobile phones (e.g., Maps on iPhone), tablets, and TV platforms and communicate with servers and websites. Examples of such platforms are Apple’s iOS,

Google's Android, and LG's Smart TV App platform. Still others are apps running on tablets such as Adobe Photoshop Touch for iPad.²² (While at the moment of writing both web-based and mobile applications have limited editing capabilities in comparison with their desktop counterparts, this may already have changed by the time you are reading this book).

The development of mobile software platforms led to the increasing importance of certain media application types (and corresponding cultural activities) such as "media uploaders" (apps designed for uploading media content to media sharing sites). To put this differently, managing media content (for example, organizing photos in Picasa) and also "meta-managing" (i.e. managing the systems which manage it such as organizing a blogroll) have become as central to a person's cultural life as creating this content.

This book is about media software—its conceptual history, the ways it redefined the practice of media design, the aesthetics of the media being created, and creators' and users' understanding of "media." How can we place media software inside other categories and also break it into smaller categories? Let us start again with our definition, which I will rephrase here. Media software are *programs that are used to create and interact with media objects and environments*. It is a subset of the larger category of "application software"—the term which is itself in the process of changing its meaning as desktop applications (applications which run on a computer) are supplemented by mobile apps (applications running on mobile devices) and web applications (applications which consist of a web client and the software running on a server). Media software enables creation, publishing, accessing, sharing, and remixing different types of media (such as images, moving image sequences, 3D shapes, characters, and spaces, text, maps, interactive elements), as well as various projects and services which use these elements. These projects can be non-interactive (2D designs, motion graphics, film shots) or interactive (media surfaces and other interactive installations). The online services are by their very nature always interactive (websites, blogs, social networks, social media services, games, wikis, web media and app stores such as Google Play and Apple iTunes, other shopping sites,

²² <http://www.adobe.com/products/mobileapps/> (March 12, 2012).

and so on)—while a user is not always given the ability to add to or modify content, s/he always navigates and interacts with the existing content using interactive interface.

Given that today the multi-billion global culture industry is enabled by media applications, it is interesting that there is no single accepted way to classify them. The Wikipedia article on “application software” includes the categories of “media development software” and “content access software” (divided into web browsers, media players, and presentation applications).²³ This is generally useful but not completely accurate—since today most “content access software” also includes at least some media editing functions. For example, the SeaMonkey browser from Mozilla Foundation includes an HTML editor;²⁴ QuickTime Player can be used to cut and paste parts of video; iPhoto supports a number of photo editing operations. Conversely, in most cases “media development” (or “content creation”) software such as Word or PowerPoint is used to both develop and access content. (This co-existence of authoring and access functions is an important distinguishing feature of software culture.) If we visit the websites of popular makers of these software applications such as Adobe and Autodesk, we will find that these companies may break their products by market (web, broadcast, architecture, and so on) or use sub-categories such as “consumer” and “pro.” This is as good as it gets—another reason why we should focus our theoretical tools on interrogating media software.

While I will focus on media applications for creating and accessing “content” (i.e. media artifacts), cultural software also includes tools and services that are specifically designed for *communication and sharing of information and knowledge*, i.e. “social software” (categories 3–4 in my list). The examples include search engines, web browsers, blog editors, email applications, instant messaging applications, wikis, social bookmarking, social networks, virtual worlds, and prediction markets. The familiar names include Facebook, the family of Google products (Google Web search, Gmail, Google Maps, Google+, etc.), Skype, MediaWiki, and Blogger. However, since at the end of the 2000s, numerous software apps and services started to include email, post,

²³ http://en.wikipedia.org/wiki/Application_software

²⁴ <http://www.seamonkey-project.org/>

and chat functions (often via a dedicated “Share” menu), to an extent, *all software became social software*.

Of course, people do not share everything online with others—at least, not yet and not everybody. Therefore, we should also include software *tools for personal information management* such as project managers, database applications, and simple text editors or note-taking apps that are included with every computer device being sold.

These and all other categories of software shift over time. For instance, during the 2000s the boundary between “personal information” and “public information” has been reconfigured as people started to routinely place their media on media sharing sites, and also communicate with others on social networks.

In fact, the whole reason behind the existence of *social media and social networking services and hosting websites* is to erase this boundary as much as possible. By encouraging users to conduct larger parts of their social and cultural lives on their sites, these services can both sell more ads to more people and ensure the continuous growth of their user base. With more of your friends using a particular service and offering more information, media, and discussions there, you are more likely to also join that service.

As many of these services began to offer more and more advanced media editing and information management tools along with their original media hosting and communication and social networking functions, they did manage to largely erase another set of boundaries (from the PC era): those between application programs, operating system, and data. Facebook in particular was very aggressive in positioning itself as a complete “social platform” which can replace various stand-alone communication programs and services.

Until the rise of social media and the proliferation of mobile media platforms, it was possible to study media production, dissemination, and consumption as separate processes. Similarly, we could usually separate production tools, distribution technologies, and media access devices and platforms—for example, the TV studio, cameras, lighting, and editing machines (production), transmission systems (distribution), and television sets (access). Social media and cloud computing in general erase these boundaries in many cases and at the same time introduce new ones (client/server, open access/commercial). The challenge of software studies is to be able to use terms such as “content” and “software

application” while always keeping in mind that the current social media/cloud computing paradigms are systematically reconfiguring the meaning of these terms.

Since creation of interactive media often involves writing some original computer code, the *programming environments* also can be considered under cultural software. Moreover, *the media interfaces* themselves—icons, folders, sounds, animations, vibrating surfaces, and touch screens—are also cultural software, since these interfaces mediate people’s interactions with media and other people. I will stop here but this list can easily be extended to include additional categories of software as well.

The interface category is particularly important for this book. I am interested in how *software appears to users*—i.e. what *functions* it offers to create, share, reuse, mix, create, manage, share and communicate content, *the interfaces* used to present these functions, and *assumptions and models about a user, her/his needs, and society* encoded in these functions and their interface design.

These functions offered by an application are embedded in application commands and tools. They define what you can do with a given app, and how you can do it. This is clear; but I need to make one important point about interfaces to avoid any confusion. Many people still think that contemporary computer devices use a Graphical User Interface (GUI). In reality, the original GUI of the early 1980s (icons, folders, menus) has been gradually extended to include other media and senses (sounds, animations, and vibration feedback which may accompany user interactions on a mobile device, voice input, multi-touch gesture interfaces, etc.) This is why the term “media interface” (used in the industry) is a more accurate description of how interfaces work today. The term accurately describes interfaces of computer operating systems such as Windows and Mac OS, and mobile OS such Android and iOS; it is even applicable to interfaces of game consoles and mobile phones, as well as interactive stores²⁵ or museum installations which use all types of media besides graphics to communicate with the users.²⁶

²⁵ For examples, see Nanika’s projects for Nokia and Diesel, <http://www.nanikawa.com/>; Audi City in London, opened 2012.

²⁶ For example, see interactive installations at the Nobel Peace Center in Oslo: Nobel Chamber, Nobel Field, and Nobel Electronic Wall Papers, <http://www.nobelpeacecenter.org/en/exhibitions/peace-prize-laureates/>

I also need to comment on the “media/content” *vs.* “data/information/knowledge” categories used to organize my list of types of cultural software above. As with many other categories that I will use in this book, I think of them as marking the two parts of the same continuous dimension rather than as being discrete either/or boxes. A feature film is a good example of the first category, and an Excel spreadsheet represents the second category—but between such clear-cut examples, there are numerous other cases which are both. For example, if I make an information visualization of the data in the spreadsheet, this visualization now fits equally into both categories. It is still “data,” but data represented in a new way which allows us to arrive at insights and “knowledge.” It also becomes a piece of visual media which appeals to our senses in the same way as photographs and paintings do.

The reason that our society places these two sets of terms in opposition has to do with the histories of the media and information industries. Modern “media” is the result of the technologies and institutions which developed between the second half of the eighteenth and first half of the twentieth centuries: large-scale newspaper, magazine and book publishing, photography, cinema, radio, television and the record industry. “Data” comes from a number of separate professional fields with distinct histories: social statistics, economics, business management, and financial markets. It is only in the beginning of the twenty-first century that data leaves professional domains to become of interest to society at large. Data becomes “sexy” and “hip,” with governments and cities creating their own data portals (for example, data.gov and data.gov.uk), visualizations of data entering exhibitions of major museums such as MOMA (*Design and Elastic Mind*, 2008), the computer “nerds” becoming heroes of Hollywood films (*Social Network*, 2010), and Google Analytics, Facebook, YouTube and Flickr all offering detailed data about your website or media sharing account. Of course, since media software operations (as well as any other computer processing of media for research, commercial or artistic purposes) are only possible because the computer represents *media* as *data* (discrete elements such as pixels, or equations defining vector graphics in vector files such as EPS), the development of media software and its adoption as the key media technology (discussed in this book) is an important contributor to the gradual coming together of media and data.

Software includes many other technologies and types, and computers and computer devices also perform lots of other functions besides creating and playing media. And of course, software needs hardware to run; and networks are also an essential part of our digital culture. Therefore, my focus on software applications for creating, editing, and playing media is likely to annoy some people. Not everybody uses Photoshop, Flash, Maya, and other applications to create media. A significant number of people work with media by writing their own computer programs and scripts, or modifying programs written by others. These are programmers responsible for the coding of websites, web applications, and other interactive applications, software artists, computer scientists working on the development of new algorithms, students using Processing and other high-level media programming languages, and other groups. All of them may ask me why I single out software in the form of consumer products (i.e. applications)—as opposed to the activity of programming? And what about the gradual democratization of software development and the gradual increase in the number of culture professionals and students who can program or write scripts? Should I not put my energy into promoting programming rather than explaining applications?

The reason for my choice is my commitment to understand the mainstream cultural practices rather than to emphasize (as many cultural critics do) the exceptions, no matter how progressive they may be. Although we do not have an exact number, I assume that the number of people who work in media and who can also program is tiny in comparison to the army of application users. Today, a typical professional graphic designer, film editor, product designer, architect, music artist—and certainly a typical person uploading videos to YouTube or adding photos and video on her/his blog—can neither write nor read software code. (Being able to read and modify HTML markup, or copy already pre-packaged lines of Javascript code is very different from programming.) Therefore, if we want to understand how software has already re-shaped media both conceptually and practically, we have to take a close look at the everyday tools used by the great majority of both professional and non-professional users—i.e. application software, web-based software, and, of course, mobile apps. (This book highlights the first category at the expense of the second and the third—because at this point, creation of professional media still

requires applications running on a laptop or desktop, often with a significant amount of RAM and large hard drives; and also because currently web-based and mobile software are still evolving quite rapidly in contrast to desktop applications such as Photoshop and Final Cut which change only incrementally from release to release).

Any definition is likely to delight some people and to annoy others. Therefore, I also would like to address another likely objection to the way I defined the term “cultural software” (with “media software” being its subset). The term “culture” is not reducible to separate media and design “objects” which may exist as files on a computer and/or as executable software programs or scripts. It includes symbols, meanings, values, language, habits, beliefs, ideologies, rituals, religion, dress and behavioral codes, and many other material and immaterial elements and dimensions. Consequently, cultural anthropologists, linguists, sociologists, and many humanists may be annoyed at what may appear as an uncritical reduction of all these dimensions to a set of tools for creating and playing media files.

Am I saying that today “culture” is equated with a particular subset of application software and the media objects and experiences that can be created with their help? Of course not. However, what I am saying—and what I hope this book explicates in more detail—is that at the end of the twentieth century humans have added a fundamentally new dimension to everything that counts as “culture.” This dimension is software in general, and application software for creating and accessing content in particular.

I am using the metaphor of a new dimension on purpose. That is, “cultural software” is not simply a new object—no matter how large and important—which has been dropped into the space which we call “culture.” Thus, it would be imprecise to think of software as simply another term which we can add to the set which includes music, visual design, built spaces, dress codes, languages, food, club cultures, corporate norms, ways of talking and using a body, and so on. And while we can certainly study “the culture of software”—programming practices, values and ideologies of programmers and software companies, the cultures of Silicon Valley and Bangalore, etc.—if we only do this, we will miss the real importance of software. Like the alphabet, mathematics, printing press, combustion engine, electricity, and integrated circuits, software re-adjusts and re-shapes everything it

is applied to—or at least, it has a potential to do this. Just as adding a new dimension adds a new coordinate to every point in space, “adding” software to culture changes the identity of everything that a culture is made from. (In this respect, software is a perfect example of what McLuhan meant when he wrote, the ‘message of any medium or technology is the change of scale or pace or pattern that it introduces into human affairs.’²⁷)

To summarize: our contemporary society can be characterized as a *software society* and our culture can be justifiably called a *software culture*—because today software plays a central role in shaping both the material elements and many of the immaterial structures that together make up “culture.”

From documents to performances

The use of software re-configures most basic social and cultural practices and makes us rethink the concepts and theories we developed to describe them. As one example of this, consider the modern “atom” of cultural creation, transmission, and memory: a “document,” i.e. some content stored in a physical form, that is delivered to consumers via physical copies (books, films, audio record), or electronic transmission (television). In software culture, we no longer have “documents,” “works,” “messages” or “recordings” in twentieth-century terms. Instead of fixed documents that could be analyzed by examining their structure and content (a typical move of the twentieth-century cultural analysis and theory, from Russian Formalism to Literary Darwinism), we now interact with dynamic “software performances.” I use the word “performance” because what we are experiencing is constructed by software in real time. So whether we are exploring a dynamic website, playing a video game, or using an app on a mobile phone to locate particular places or friends nearby, we are engaging not with pre-defined static documents but with the dynamic outputs of a real-time computation happening on our

²⁷ Marshall McLuhan, *Understanding Media: The Extensions of Man* (New York: McGraw Hill, 1964), quoted in *New Media Reader*, Noah Wardrip-Fruin and Nick Montfort (eds) (The MIT Press, 2003), p. 203.

device and/or the server. Computer programs can use a variety of components to create these performances: design templates, files stored on a local machine, media from the databases on the network server, the real-time input from a mouse, touch screen, joystick, our moving bodies, or some other interface. Therefore, although some static documents may be involved, the final *media experience constructed by software usually does not correspond to any single static document stored in some media*. In other words, in contrast to paintings, literary works, music scores, films, industrial designs, or buildings, a critic cannot simply consult a single “file” containing all of the work’s content.

Even in such seemingly simple cases as viewing a PDF document or opening a photo in a media player, we are already dealing with “software performances”—since it is software which defines the options for navigating, editing, and sharing the document, rather than the document itself. Therefore examining the PDF file or a JPEG file the way twentieth-century critics would examine a novel, a movie, or a TV program will only tell us some things about the experience we get when we interact with this document via software—but not everything. This experience is equally shaped by the interface and the tools provided by software. This is why the examination of the tools, interfaces, assumptions, concepts, and the history of cultural software—including the theories of its inventors who in the 1960s and 1970s have defined most of these concepts—is essential if we are to make sense of contemporary media.

This shift in the nature of what constitutes a media “document” also calls into question well-established cultural theories that depend on this concept. Consider the intellectual paradigm that dominated the study of media since the 1950s—the “transmission” view of culture developed in Communication Studies. Communication scholars have taken the model of information transmission formulated by Claude Shannon in his 1948 article *A Mathematical Theory of Communication* (1948)²⁸ and his subsequent book published with Warren Weaver in 1949,²⁹ and applied

²⁸ C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, 623–56, July, October, 1948, <http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf>

²⁹ Claude E. Shannon and Warren Weaver, *The Mathematical Theory of Communication* (University of Illinois Press, 1949).

its basic model of communication to mass media. The paradigm described mass communication (and sometimes culture in general) as a communication process between the authors who create and send messages and the audiences that receive them. According to this paradigm, the messages were not always fully decoded by the audiences for technical reasons (noise in transmission) or semantic reasons (they misunderstood the intended meanings).

Classical communication theory and media industries considered such partial reception a problem; in contrast, in his influential 1980 article “Encoding/decoding”³⁰ the founder of British Cultural Studies, Stuart Hall, argued that the same phenomenon is positive. Hall proposed that the audiences construct their own meanings from the information they receive. Rather than being a communication failure, the new meanings are active acts of intentional reinterpretation of the sent messages. But both the classical communication studies and cultural studies implicitly took for granted that the message was something complete and definite—regardless of whether it was stored in physical media (e.g. magnetic tape) or created in real time by a sender (a live TV broadcast). Thus, the receiver of communication was assumed to read all of the advertising copy, see a whole movie, or listen to the whole song and only after that s/he would interpret it, misinterpret it, assign his/her own meanings, appropriate it, remix it, etc.

While this assumption has already been challenged by the introduction of the DVR (digital video recorder) in 1999, which led to the phenomenon of *time shifting*, it simply does not apply to interactive software-driven media. The interfaces of media access applications, such as web browsers and search engines, the hyper-linked architecture of the World Wide Web, and the interfaces of particular online media services offering massive numbers of media artifacts for playback preview and/or purchase (Amazon, Google Play, iTunes, Rhapsody, Netflix, etc.), encourage people to “browse,” quickly moving both *horizontally* between media (from one search result to the next, from one song to another, etc.) and vertically, *through* the media artifacts (e.g., from the contents listing of a music album to a particular track). They also made it easy to start playing/viewing media at an arbitrary point, and to

³⁰ Stuart Hall, “Encoding/decoding,” in *Culture, Media, Language*, ed., Centre for Contemporary Cultural Studies (London: Hutchinson, 1980).

leave it at any point. In other words, the “message” that the user “receives” is not just actively “constructed” by him/her (through a cognitive interpretation) but also actively managed (defining what information s/he is receiving and how).

It is at least as important that when a user interacts with a software application that presents media content, this content often does not have any fixed finite boundaries. For instance, a user of Google Earth is likely to experience a different “Earth” every time s/he is accessing the application. Google could have updated some of the satellite photographs or added new Street Views; new 3D buildings, new layers, and new information on already existing layers were also likely to be added. Moreover, at any time a user of the application can load more geospatial data created by other users and companies by either selecting one of the options in the Add menu (Google Earth 6.2.1 interface), or directly opening a KLM file. Google Earth is a typical example of a new type of media enabled by the web—an interactive document which does not have all of its content pre-defined. Its content changes and grows over time.

In some cases this may not affect in any significant way the larger “messages” “communicated” by the software application, web service, game, or other type of interactive media. For example, Google Earth’s built-in cartographic convention of representing the Earth using the General Perspective Projection (a particular map projection method of cartography) does not change when users add new content and turn on and off map layers. The “message” of this representation is always present.³¹

However, since a user of Google Earth can also add his/her own media and information to the base representation provided by the application, creating complex and media rich projects on top of existing geoinformation, Google Earth is not just a “message.” It is a *platform* for users to build on. And while we can find some continuity here with the users’ creative reworking of commercial media in the twentieth century—pop art and appropriation, music remixes, slash fiction and video,³² and so on, the differences are larger than the similarities.

³¹ http://en.wikipedia.org/wiki/Google_earth#Technical_specifications (March 14, 2012).

³² See, for instance, Constance Penley, “Feminism, Psychoanalysis, and the Study of Popular Culture,” in *Cultural Studies*, ed. Lawrence Grossberg (Routledge, 1992).

This shift from messages to platforms was in the center of the Web's transformation around 2004–6. The result was named Web 2.0. The 1990s websites presenting particular content created by others (and thus, communicating “messages”) were supplemented by social networks and social media sites where the users can share, comment on, and tag their own media. The Wikipedia article on Web 2.0 describes these differences as follows: “A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as creators (prosumers) of user-generated content in a virtual community, in contrast to websites where users (consumers) are limited to the passive viewing of content that was created for them. Examples of Web 2.0 include social networking sites, blogs, wikis, video sharing sites, hosted services, web applications, mashups and folksonomies.”³³ For example, to continue with the Google Earth example, users added many types of global awareness information, including fair trade certification, Greenpeace data, and United Nations Millennium Development Goals Monitor.³⁴ In another example, you can incorporate Google Maps, Wikipedia, or content provided by most other large web 2.0 sites directly in your web mashup—an even more direct way of taking the content provided by web services and using it to craft your own custom platforms.

The wide adoption of Web 2.0 services along with various web-based communication tools (online discussion forums about all popular software, collaborative editing on Wikipedia, Twitter, etc.) enables quick identifications of omissions, selections, censorship and other types of “bad behavior” by software publishers—another feature which separates content distributed by web-based companies from mass media of the twentieth century. For example, every article on Wikipedia about a Web 2.0 service includes a special section about controversies, criticism, or errors.

In many cases, people can also use alternative open source equivalents of paid and locked applications. Open source and/or free software (not all free software is open source) often allow for additional ways of creating, remixing and sharing both content and new software additions. (This does not mean

³³ http://en.wikipedia.org/wiki/Web_2.0 (March 14, 2012).

³⁴ http://en.wikipedia.org/wiki/Google_earth (March 14, 2012).

that open source software always uses different assumptions and technologies than the commercial software.) For example, one can choose to use a number of alternatives to Google Maps and Google Earth—OpenStreetMap, Geocommons, WorldMap, and others which all have open source or free software licenses.³⁵ (Interestingly, commercial companies also often use data from such free collaboratively created systems because they contain more information than the companies' own systems. OpenStreetMap, which by early 2011 had 340,000 contributors,³⁶ is used by Flickr and Foursquare.³⁷) A user can also examine the code of open-source software to fully understand its assumptions and key technologies.

Continuously changing and growing content of web services and sites; variety of mechanism for navigation and interaction; the abilities to add one's own content and mashup content from various sources together; architectures for collaborative authoring and editing; mechanisms for monitoring the providers—all these mechanisms clearly separate interactive networked software-driven media from twentieth-century media documents. But even when a user is working with a single local media document that is stored in a single computer file (a rather rare situation these days), such a document mediated through software interface has a different identity from a twentieth-century media document. The user's experience is only partly defined by the file's content and its organization. The user is free to navigate the document, choosing both what information to see and the sequence in which s/he is seeing it. And while "old media" (with the exception of twentieth-century broadcasting) also provided this random access, the interfaces of software-driven media players/viewers provide many additional ways for browsing media and selecting what and how to access.

For example, Adobe Acrobat can display thumbnails of every page in a PDF document; Google Earth can quickly zoom in and out from the current view; online digital libraries, databases and repositories containing scientific articles and abstracts such as the ACM

³⁵ <http://geocommons.com>, <http://www.openstreetmap.org>, <http://worldmap.harvard.edu>

³⁶ <http://en.wikipedia.org/wiki/Counter-mapping#OpenStreetMap> (March 27, 2012).

³⁷ http://en.wikipedia.org/wiki/OpenStreetMap#Derivations_of_OpenStreetMap_Data (March 27, 2012).

Digital Library, IEEE Xplore, PubMed, Science Direct, SciVerse Scopus, and Web of Science show articles which contain references to the one you currently selected. Most importantly, these new tools and interfaces are *not hard-wired* to the media documents themselves (such as a random access capacity of a printed book) or media access machines (such as a radio); instead they are part of the separate *software* layer. This media architecture enables easy addition of new navigation and management tools without any change to the documents themselves. For instance, with a single click, I can add sharing buttons to my blog, thus enabling new ways of circulation for its content. When I open a text document in Mac OS X Preview media viewer, I can highlight, add comments and links, draw, and add thought bubbles. Photoshop allows me to save my edits on separate “adjustment layers,” without modifying the original image. And so on.

Why the history of cultural software does not exist

“Всякое описание мира сильно отстаёт от его развития.”

(Translation from Russian: “Every description of the world substantially lags behind its actual development.”)

Тая Катюша, VJ on MTV.ru, 2008.³⁸

We live in a software culture—that is, a culture where the production, distribution, and reception of most content is mediated by software. And yet, most creative professionals do not know anything about the intellectual history of software they use daily—be it Photoshop, Illustrator, GIMP, Final Cut, After Effects, Blender, Flame, Maya, MAX, or Dreamweaver.

Where does contemporary cultural software come from? How were its metaphors and techniques arrived at? And why was it developed in the first place? Currently most prominent computer and web companies have been extensively covered in media, so their history is relatively well-known (for instance, Facebook, Google, and Apple). But this is only the tip of the iceberg. The history of media authoring and editing software remains pretty much unknown. Despite the common statements that the

digital revolution is at least as important as the invention of the printing press, we are largely ignorant of how the key part of this revolution—i.e., media software—was invented. When you think about this, it is unbelievable. People in the business of culture know about Gutenberg (printing press), Brunelleschi (perspective), The Lumière Brothers, Griffith and Eisenstein (cinema), Le Corbusier (modern architecture), Isadora Duncan (modern dance), and Saul Bass (motion graphics). (If you happen not to know one of these names, I am sure that you have other cultural friends who do). And yet, even today, relatively few people have heard of J. C. R. Licklider, Ivan Sutherland, Ted Nelson, Douglas Engelbart, Alan Kay, and their collaborators who, between approximately 1960 and 1978, gradually turned the computer into the cultural machine it is today.

Remarkably, the history of cultural software as a discrete category does not yet exist. What we have are a number of largely biographical books about some of the key individual figures, and research labs such as Xerox PARC or MIT Media Lab—but no comprehensive synthesis which would trace the genealogical tree of media tools. And we also do not have any detailed studies which would relate the history of cultural software to the history of media, media theory, or history of visual culture.

Modern art institutions—museums such as the MOMA and the Tate, art book publishers such as Phaidon and Rizzoli, etc.—promote the history of modern art. Hollywood is similarly proud of its own history—the stars, the directors, the cinematographers, and the classical films. So how can we understand the neglect of the history of cultural computing by our cultural institutions and computer industry itself? Why, for instance, does Silicon Valley not have a museum for cultural software? (The Computer History museum in Mountain View, California has an extensive permanent exhibition which is focused on hardware, operating systems, and programming languages—but not on the history of software.³⁸)

I believe that the major reason has to do with economics. Originally misunderstood and ridiculed, modern art has eventually become a legitimate investment category—in fact, by middle of the 2000s, the paintings of a number of twentieth-century artists

³⁸ <http://www.mtv.ru/air/vjs/taya/main.wbp> (February 21, 2008).

were selling for more money than the works of the most famous classical artists. Similarly, Hollywood continues to receive profits from old movies as it reissues them in new formats (VHS, DVD, HD, Blu-ray disks, etc). What about the IT industry? It does not derive any profits from the old software—and therefore it does nothing to promote its history. Of course, contemporary versions of Microsoft Word, Adobe Photoshop, Autodesk AutoCAD, and many other popular cultural applications were built on the first versions, which often date from the 1980s, and the companies continue to benefit from the patents they filed for new technologies used in these original versions—but, in contrast to the video games from the 1980s, these early software versions are not treated as separate products which can be re-issued today. (In principle, I can imagine the software industry creating a whole new market for old software versions or applications which at some point were quite important but no longer exist today—for instance, Aldus Pagemaker. In fact, given that consumer culture systematically exploits adults' nostalgia for the cultural experiences of their teenage and youth years, it is actually surprising that early software versions were not seen as a market opportunity. If I used MacWrite and MacPaint daily in the middle of the 1980s, or Photoshop 1.0 and 2.0 in 1990–3, I think these experiences would be as much part of my “cultural genealogy” as the movies and art I saw at that time. Although I am not necessarily advocating the creation of yet another category of commercial products, if early software was widely available in simulation, it would catalyze cultural interest in software similar to the way in which wide availability of early computer games, recreated for contemporary mobile platforms, fuels the field of video game studies.

Since most theorists so far have not considered cultural software as a subject of its own, distinct from “social media,” “social networks,” “new media,” media art,” “the internet,” “interactivity,” and “cyberculture,” we lack not only a conceptual history of media editing software but also systematic investigations of *the roles of software in media production*. For instance, how did the adoption of the popular animation and compositing application After Effects in the 1990s reshape the language of moving images? How did the adoption of Alias, Maya and other 3D packages by architectural students and young architects in the same decade similarly influence the language of architecture? What about the

co-evolution of Web design tools and the aesthetics of websites—from the bare-bones HTML in 1994 to visually rich Flash-driven sites five years later, and responsive web design in the early 2010s? You will find frequent mentions and short discussions of these and similar questions in articles and conference talks, but as far as I know, there has been no book-length study about any of these subjects. Often, books on architecture, motion graphics, graphic design and other design fields will briefly discuss the importance of software tools in facilitating new possibilities and opportunities, but these discussions are not usually further developed.

In summary, a systematic examination of the connections between the workings of contemporary media software and the new communication languages in design and media (including graphic design, web design, product design, motion graphics, animation, and cinema) has not yet been undertaken. Although this book alone cannot do it all, I hope that it will provide some general models of how such connections can be teased out—as well as provide a detailed analysis of how software use redefined certain cultural areas (e.g., motion graphics and visual design).

By focusing on the theory of software for media design, this book aims to complement the work of a few other theorists that have already examined software responsible for game platforms and design (Ian Bogost, Nick Montfort), and electronic literature (Noah Wardrip-Fruin, Matthew Kirschenbaum).

In this respect, the related fields of code studies and platform studies being developed by Mark Marino,²⁴ Nick Montfort, Ian Bogost and others are playing a very important role. According to Marino (and I completely agree), the three fields of software studies, code studies, and game studies complement each other: “Critical code studies is an emerging field related to software studies and platform studies, but it’s more closely attuned to the code itself of a program rather than the program’s interface and usability (as in software studies) or its underlying hardware (as in platform studies).”³⁹

³⁹ <http://chnm2011.thatcamp.org/05/24/session-proposal-critical-codestudies/> (July 14, 2011).

Summary of the book's narrative

Between the early 1990s and the middle of the 2000s, media software has replaced most of the other media technologies that emerged in the nineteenth and twentieth centuries. Most contemporary media is created and accessed via cultural software—and yet, surprisingly, few people know about its history. What was the thinking and motivation of people who between 1960 and the late 1970s created the concepts and practical techniques that underlie today's cultural software? How does the shift to software-based production methods in the 1990s change our concepts of “media”? How have interfaces and the tools of content development software reshaped and continued to shape the aesthetics and visual languages we see in contemporary design and media? These are the key questions that I take up in this book.

My aim is not provide a comprehensive history of cultural software in general, or media authoring software in particular. Nor do I aim to discuss all the new creative techniques media software enables across dozens of cultural fields. Instead, I will trace a *particular path through this history* that will take us from 1960 to today and which will pass through some of its most crucial points. In the following I summarize this narrative and also introduce some of the key concepts developed in each part of the book.

Part 1 looks at the 1960s and 1970s. While new media theorists have spent considerable efforts in trying to understand the relationships between digital media and older physical and electronic media, the important sources—the writing and projects by Ivan Sutherland, Douglas Engelbart, Ted Nelson, Alan Kay, and other pioneers of cultural software working in these decades—still remain largely unexamined. What were their reasons for inventing the concepts and techniques that today make it possible for computers to represent, or “remediate” other media? Why did these people and their colleagues work to systematically turn a computer into a machine for media creation and manipulation? These are the questions that I take in Part 1, which explores them by focusing on the ideas and work of the key protagonist of “cultural software movement”—Alan Kay. (It is certainly possible to construct a more exclusive or an alternative history which will pay equal attention to dozens of brilliant people who worked with these people and

who, together, invented all the details which form the DNA of contemporary media software—for instance, Bob Taylor, Charles Thacker, John Warnock, and others working at Xerox PARC in the 1970s; or the people who contributed to the design of the first Macintosh.⁴⁰ However, since we do not yet even have a theoretical analysis of how the ideas of the most well-known figures of the 1960s collectively changed media, this book will start with these figures, and the analysis of their theoretical writings.)

I suggest that Kay and cultural software pioneers aimed to create a particular kind of new media—rather than merely simulating the appearances of old ones. These new media use already existing representational formats as their building blocks, while adding many previously nonexistent properties. At the same time, as envisioned by Kay, these media are expandable—that is, users themselves should be able to easily add new properties, as well as to invent new media. Accordingly, Kay calls computers the first *metamedium* whose content is “a wide range of already-existing and not-yet-invented media.”

The foundations necessary for the existence of such metamedium were established between the 1960s and the late 1970s. During this period, most previously available physical and electronic media were systematically simulated in software, and a number of new media were also invented. This development takes us from the very interactive design program—Ivan Sutherland’s Sketchpad (1962)—to the commercial desktop applications that made software-based media authoring and design widely available to members of different creative professions and, eventually, media consumers as well—AutoCAD (1982), Word (1984), PageMaker (1985), Alias (1985), Illustrator (1987), Director (1987), Photoshop (1989), After Effects (1993), and others. (These PC applications were paralleled by much more expensive systems for professional markets such as the TV and video industries which got Paintbox in 1981, Harry in 1985, Avid in 1989, and Flame in 1992.)

So what happens next? Did Kay’s theoretical formulations as articulated in 1977 accurately predict the developments of the next thirty years, or have there been new developments that his concept of “metamedium” did not account for? Today we do

⁴⁰ For the stories that document the inventions by dozens of people of the multiple technologies that made up the original Macintosh, see www.folklore.com

indeed use a variety of previously existing media simulated in software as well as new previously non-existent media types. Both have been continuously extended with new properties. Do these processes of invention and amplification take place at random, or do they follow particular paths? In other words, what are the key mechanisms responsible for the extension of the computer metamedium?

Parts 2 and 3 are devoted to these questions. They look at the number of different mechanisms which drove development and expansion of the computer metamedium, with the focus on the 1990s when media software was gradually adopted in all areas of professional media production. I use three different concepts to describe these developments and the new aesthetics of visual media which developed in the second part of the 1990s after the processes of adoption reached sufficient speed. These three concepts are *media hybridization*, *evolution*, and *deep remix*. Part 2 develops the theoretical analysis of this second stage of metamedium development, illustrating it with a number of examples drawn from different genres of digital media. Part 3 focuses in detail on the use of software for visual design (motion graphics and graphics design), analyzing the relationships between the new aesthetics of moving and still images and compositions, and the operations and interfaces of software used to create them such as After Effects.

I argue that in the process of the translation from physical and electronic media technologies to software, all individual techniques and tools that were previously unique to different media “met” within the same software environment. This meeting had fundamental consequences for human cultural development and for the media evolution. It disrupted and transformed the whole landscape of media technologies, the creative professions that use them, and the very concept of media itself.

Once they were simulated in a computer, previously incompatible techniques of different media begin to be combined in endless new ways, leading to new media hybrids, or, to use a biological metaphor, new “media species.” As just one example among countless others, think, for instance, of the popular Google Earth application, combining techniques of traditional mapping, the concepts from the field of Geographical Information Systems (GIS), 3D computer graphics and animation, social software, search, and other elements and functions. In my view, this *ability to*

combine previously separate media techniques represents a fundamentally new stage in the history of human media, human semiosis, and human communication, enabled by its “softwarization.”

I describe this new stage in media evolution using the concept of *hybridity*. In the first stage, most existing media were simulated in a computer and a number of new types of media that can only be realized in a computer were invented. In the second stage, these simulated and new mediums started exchanging properties and techniques.

To distinguish these processes from more familiar remixes, I introduce the new term *deep remixability*. Normally a remix is a combination of content from a single medium (like in music remixes), or from a few mediums (like Anime Music Video works which combine content from anime and music video). However, the software production environment allows designers to remix not only the content of different media types, but also their fundamental techniques, working methods, and ways of representation and expression.

While today hybridization and deep remix can be found at work in all areas of culture where software is used, I focus on particular area to demonstrate how it functions in detail. This area is *visual design* in general, and *motion graphics* in particular. Motion graphics is a dynamic part of contemporary culture, which, as far as I know, has not yet been theoretically analyzed in detail anywhere. Although selected precedents for contemporary motion graphics can already be found in the 1950s and 1960s in the works by Saul Bass and Pablo Ferro, its exponential growth from the middle of the 1990s is directly related to the adoption of software for moving image design—specifically, After Effects software released by Adobe in 1993. Deep remixability is central to the aesthetics of motion graphics. That is, the larger proportion of motion graphics projects done today around the world derive their aesthetic effects from combining different techniques and media traditions—animation, drawing, typography photography, 3D graphics, video, etc.—in new ways. As a part of my analysis, I look at how the typical software-based production workflow in a contemporary design studio—the ways in which a project moves from one software application to another—shapes the aesthetics of motion graphics, and visual design in general.

The next major wave of computerization of culture has to do with different types of software—social networks, social media

services, and apps for mobile platforms. The wave of social networks and social media started slowly, erupted in 2005–2006 (Flickr, YouTube) and continues to move forward and expand its reach. The 1990s’ media revolution impacted *professional creatives*; the 2000s’ media revolution affected *the rest of us*—i.e. the hundreds of millions who use Facebook, Twitter, Firefox, Safari, Google Search and Maps, Flickr, Picasa, Vimeo, Blogger, and numerous apps and services available on mobile platforms.

Because we are still in the middle of social media diffusion, with some popular social media services going out of favor and others gaining speed (for example, think of the fate of MySpace), and the “social” functionality of software still expanding, I decided that offering the detailed theoretical analysis of this new wave would be premature. (This became clear after I started editing the part about social media which I originally had in the first book draft, and realized that some of the social media services I was analyzing in detail no longer exist... .) Instead, I am focusing on tracing the fundamental developments which made possible and shaped “digital media” before its social explosion: the ideas about the computer as a machine for media generation and editing of the 1960s–1970s, their implementation in the media applications in the 1980s–1990s, and the transformation of visual media languages which quickly followed.

To be more precise, we can frame this history between 1961 and 1999. In 1961, Ivan Sutherland at MIT designed Sketchpad, which became the first computer design system shown to the public. In 1999, After Effects 4.0 introduced Premiere import,⁴¹ Photoshop 5.5 added vector shapes,⁴² and Apple showed the first version of Final Cut Pro⁴³—in short, the current paradigm of interoperable media authoring and editing tools capable of creating professional media without special hardware beyond the off-the-shelf computer was finalized. And while professional media tools continued to evolve after this period, the changes so far have been incremental. Similarly, the languages of professional visual media created with this software did not change significantly after their radical transformation in the second part of the 1990s.

⁴¹ http://en.wikipedia.org/wiki/After_Effects#History (July 7, 2011).

⁴² http://en.wikipedia.org/wiki/Adobe_Photoshop_release_history (July 7, 2011).

⁴³ http://en.wikipedia.org/wiki/Final_Cut_Pro#History (July 7, 2011).

To illustrate this continuity, the examples of particular media projects that I will analyze will be drawn from both the 1990s and the 2000s. However, when discussing interfaces and commands of media applications, I will use the recent versions in order to make the discussion as relevant as possible for the software users. Accordingly, I will also take into account the addition of social media capacities in all consumer-level media software that took place at the end of the 2000s (e.g., the “share” menu in iPhoto). And because the hybridization mechanism is not limited to professional media software and professionally created media, but also plays the key role in evolution of social web software and services, I will include prominent examples of such services, for example, Google Earth.

I also need to comment on my choices of particular media software applications as examples. I have chosen to focus on the desktop applications for media authoring most widely used today—Photoshop, Illustrator, InDesign, Dreamweaver, After Effects, Final Cut, Maya, 3ds Max, Word, PowerPoint, etc. These programs exemplify different categories of media authoring software: image editing, vector graphics, page layout, web design, motion graphics, video editing, 3D modeling and animation, word processing, and presentation. I will also be making references to popular web browsers (Firefox, Chrome, Internet Explorer), blogging tools and publishing services (WordPress, Blogger), social networks (Facebook, Twitter, Google+), media sharing services (Flickr, Pinterest, YouTube, Vimeo), email services and clients (Gmail, Microsoft Outlook), web-based office suites (Google Docs), and consumer geographic information systems (Google Earth, Bing Maps). Since I am interested in how users interact with media, another key software category for this book is media players pre-installed on new computers (Windows Media Player, iTunes, QuickTime) and document viewing applications (Adobe Reader, Mac OS Preview). As some programs and web services become less popular and new ones gain market share, the list above may look somewhat different by the time you are reading this, and many applications may also fully migrate from desktop to the web—but the categories are likely to remain the same.

Because I want to make my discussions as relevant as possible to contemporary designers and artists, the names of historically important programs which are no longer popular or do not exist

will only be mentioned in passing. Examples are QuarkXPress, WordPerfect, and Macromedia Director. Luckily for us, the two programs that I will analyze in detail—Photoshop (Chapter 2) and After Effects (Chapter 5)—are as popular today as they were in the 1990s.

I will not be discussing other types of digital media authoring and editing systems which were quite important in the 1980s and 1990s. Because during this period graphics capacities of personal computers were still limited, these systems ran on graphics workstations (specialized minicomputers) from Silicon Graphics or used proprietary hardware. Here are examples listed in chronological order, with the function of the system, company name and year of the first release appearing in parentheses: Paintbox (graphics for broadcast television, Quantel, 1981), Mirage (digital real-time video effects processor, Quantel, 1982),⁴⁴ Personal Visualizer (3D modeling and animation, Wavefront, 1988), Henry and Hal (effects editor and graphics and compositing systems, Quantel, 1992) Inferno and Flame (compositing for film and video, Discreet Logic, 1992).

In the middle 1990s, Flame together with the SGI workstation cost \$450,000; an Inferno system cost \$700,000.⁴⁵ Inferno 5 and Flame 8 introduced in 2003 had suggested list prices of \$571,500 and \$266,500, respectively.⁴⁶ Because of these prices, such systems were only used in television and film studios or in big video effects companies.

Today the most demanding areas of media production which involve working with massive amounts of data—feature films, feature animations, TV commercials—still rely on these systems' expensive software. While at the end of the 2000s the companies started to offer versions of these programs for PC, Macs, and Linux, today the highest-end versions still often require special hardware, and their prices are still quite high. (For example, the 2010 edition of Autodesk Flame Premium, a suite containing Smoke, Flame and

⁴⁴ http://en.wikipedia.org/wiki/Quantel_Mirage (August 23, 2012).

⁴⁵ "Discreet Logic Inc. History," <http://www.fundinguniverse.com/company-histories/discreet-logic-inc-history/>

⁴⁶ Autodesk, "Discreet Delivers inferno 5, flame 8 and flint 8," January 23, 2003, <http://investors.autodesk.com/phoenix.zhtml?c=117861&p=irol-newsArticle&ID=374002>

Lustre used for video editing, effects, and color grading was offered for \$125,000.)

Because I do not expect a typical reader of this book to have a working experience with these expensive systems, I will not be referring to them further in this book. However, a more comprehensive history of the moving image media of the 1980s–1990s (which I hope somebody will write in the future) will definitely need to do an archeology and genealogy of these systems and their use.

Finally, one more explanation is in order. Some readers will be annoyed that I focus on commercial applications for media authoring and editing, as opposed to their open source alternatives. For instance, I discuss Photoshop rather than Gimp, and Illustrator rather than Inkscape. I love and support open source and free access, and use it for all my work. Starting in 1994, I was making all my articles available for free download on my website manovich.net. And when in 2007 I set up a research lab (www.softwarestudies.com) to start analyzing and visualizing massive large media datasets, we decided to also follow a free software/open source strategy, making the tools we develop freely available and allowing others to modify them.⁴⁷

The reason this book focuses on commercial media authoring and editing software rather than its open source equivalents is simple. In almost all areas of software culture, people use free applications and web services. The examples include web browsers, web email, social networks, apps available for mobile devices, and programming and scripting languages. The companies are not charging for these free applications and services because they are making money in other ways (advertising, charging for extra features and services, membership fees, selling devices). However, in the case of professional tools for media authoring and editing, commercial software dominates. It is not necessarily better, but it is simply used by many more people. (For example, entering “Photoshop” and “Gimp” into Google Trends shows that since 2004, the number of searches for the former is about eight times bigger than for the latter.) Since I am interested in describing the common user experiences, and the features of media aesthetics

⁴⁷ <http://lab.softwarestudies.com/p/software-for-digital-humanities.html>

common to millions of works created with the most common authoring tools that are all commercial products, these are the products I choose to analyze. And when I analyze tools for media access and collaboration, I similarly choose the most popular products—which in this case includes both free software and services provided by commercial companies (Safari, Google Earth), and free open source software (Firefox).