Lev Manovich
SOFTWARE TAKES COMMAND

THIS VERSION:
November 20, 2008.
Please note that this version has not been proofread yet, and it is also missing illustrations.
Length: 82,071 Words (including footnotes).

CREATIVE COMMONS LICENSE:
Software Takes Command by Lev Manovich is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 United States License.

ABOUT THE VERSIONS:
One of the advantages of online distribution which I can control is that I don’t have to permanently fix the book’s contents. Like contemporary software and web services, the book can change as often as I like, with new “features” and “big fixes” added periodically. I plan to take advantage of these possibilities. From time to time, I will be adding new material and making changes and corrections to the text.

LATEST VERSION:
Check www.softwarestudies.com/softbook for the latest version of the book.

SUGGESTIONS, CORRECTIONS AND COMMENTS:
send to manovich@ucsd.edu
with the word “softbook” in the email header.
Introduction: Software Studies for Beginners

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 goods or processing physical matter. Today, however, the lists of best-recognized global brands are topped with the names such as Google, Yahoo, and Microsoft. (In fact, Google was number one in the world in 2007 in terms of brand recognition.) And, at least in the U.S., the most widely read newspapers and magazines - New York Times, USA Today, Business Week, etc. - daily feature news and stories about YouTube, MySpace, Facebook, Apple, Google, and other IT companies.

What about other media? If you access CNN web site and navigate to the business section, you will see a market data for just ten companies and indexes displayed right on the home page.\(^1\) Although the list changes daily, it is always likely to include some of the same IT brands. Let's take January 21, 2008 as an example. On that day CNN list consisted from the following companies and indexes: Google, Apple, S&P 500 Index, Nasdaq Composite Index, Dow Jones Industrial Average, Cisco Systems, General Electric, General Motors, Ford, Intel.\(^2\)

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. Next we have 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 appears to be in the business of information, while the second is making consumer electronics: laptops, monitors, music players, etc. But actually, they are both really 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 daily appear in news - Yahoo, Facebook, Amazon, eBay – 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 others to write new software – Facebook, Windows, Unix, Android – 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 (media and information, as well as human interactions around these media and information) – is only the visible part of a much larger software universe.

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 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 including 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" - all these key economic players of information society can't exist without software. Data visualization software used by a scientist, spreadsheet software used a financial analyst, Web design software used by a designer working for a transnational advertising energy, 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 have by now seem to cover all aspects of IT revolution, creating a number of new disciplines such as cyber culture, Internet studies, new media theory, and digital culture, the underlying engine which drives most of these subjects—software—has received little or not direct attention. Software is still invisible to most academics, artists, and cultural professionals interested in IT and its cultural and social effects. (One important exception is Open Source movement and related issues around copyright and IP that has been extensively discussed in many academic disciplines). But if we limit critical discussions to the notions of "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” – regardless of which new feature of contemporary existence a particular social theory has focused on, all these new features are enabled by software. It is time we focus on software itself.

**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 own book *The Language of New Media* (completed in 1999; published by MIT Press in 2001), 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 both the role of software in forming contemporary culture, and cultural, social, and economic forces that are shaping 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 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 key artists active in the same historical periods, the Reader demonstrated that both belonged to the same larger epistemes. That is, often the same idea was simultaneously articulated in thinking of both artists and scientists who were inventing cultural computing. For instance, the anthology opens with the story by Jorge Borges (1941) and the article by Vannevar Bush (1945) which both contain the idea of a massive branching structure as a better way to organize data and to represent human experience.

In February 2006 Mathew Fuller who 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.”

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. At the moment of this writing (Spring 2008), software studies is a new paradigm for intellectual inquiry that is now just beginning to emerge. The MIT Press is publishing the very first book that has this term in its
title later this year (Software Studies: A Lexicon, edited by Matthew Fuller.) At the same time, a number of already published works by the leading media theorists of our times - Katherine Hayles, Friedrich A. Kittler, Lawrence Lessig, Manual Castells, Alex Galloway, and others - can be retroactively identified as belonging to "software studies." Therefore, I strongly believe that this paradigm has already existed for a number of years but it has not been explicitly named so far. (In other words, the state of "software studies" is similar to where "new media" was in the early 1990s.)

In his introduction to 2006 Rotterdam workshop Fuller writes 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.” Given that 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, object-network theory, social semiotics, or media archeology.

I think 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 can't 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, humanities, science and technology studies, and so on – 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 a new methodology. That is, it helps to practice what one writes about. It is not accidental that the intellectuals who have most systematically written about software’s roles in society and culture so far all either have programmed themselves or have been systematically involved in cultural projects which centrally involve writing of new software: Katherine Hales, Mathew Fuller, Alexander Galloway, Ian Bogust, Geet Lovink, Paul D. Miller, Peter Lunenfeld, Katie Salen, Eric Zimmerman, Matthew Kirschenbaum, William J. Mitchell, Bruce Sterling, etc. In contrast, the scholars without this experience such as Jay Bolter, Siegfried Zielinski, Manual Castells, and Bruno Latour as have not included considerations of software in their otherwise highly influential accounts of modern media and technology.

In the present decade, 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 to 1999 when I first mentioned “software studies” in *The Language of New Media*. Outside of culture and academic industries, many more people today are writing software as well. To a significant extent, this is the result of new programming and scripting languages such as JavaScript, ActionScript, PHP, Processing, and others. Another important factor is the publication of their APIs by all major Web 2.0 companies in the middle of 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 web sites.) These programming and scripting languages and APIs did not necessary make programming itself any easier. Rather, they made it much more efficient. For instance, when a young designer can create an interesting design with only couple of dozens 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.

In a 2006 article that reviewed other examples of new technologies that allow people with very little or no programming experience to create new custom software (such as Ning and Coghead), Martin LaMonica wrote about a future possibility of "a long tail for apps." Clearly, today the consumer technologies for capturing and editing media are much easier to use than even most high-level programming and scripting languages. But it does not necessary have to stay this way. Think, for instance, of what it took to set up a photo studio and take photographs in 1850s versus simply pressing a single button on a digital camera or a mobile phone in 2000s. Clearly, we are very far from such simplicity in programming. But I don't see any logical reasons why programming can't one day become as 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 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 the 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." Kittler himself programs in an assembler language - which probably determined his distrust of Graphical User Interfaces and modern software applications, which use these interfaces. In a
classical modernist move, Kittler argued that we need to focus on the “essence” of computer - which for Kittler meant mathematical and logical foundations of modern computer 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 a teacher. This practical engagement begins in the early 1980s, which was the decade of procedural programming (Pascal), rather than assembly programming. It was also the decade that saw introduction of PCs and first major cultural impact of computing as desktop publishing become popular and hypertext started to be discussed 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 1983-1984 on Apple IIE. In 1984 I saw Graphical User Interface in its first successful commercial implementation on Apple Macintosh. The same year I got the job at one of the first computer animation companies (Digital Effects) where I learned how to program 3D computer models and animations. In 1986 I was writing computer programs, which would automatically process photographs to make them look like paintings. In January 1987 Adobe Systems shipped illustrator, followed by Photoshop in 1989. The same year saw the release by The Abyss directed by James Cameron. This movie used pioneering CGI to create the first complex virtual character. And, by Christmas of 1990s, Tim Berners-Lee already created all the components of World Wide Web as it exists today: a web server, web pages, and a web browser.

In short, during one decade a computer moved from being a culturally invisible technology to being the new engine of culture. While the progress in hardware and Moore’s Law of course played crucial roles in this, even more crucial was the release of software aimed at non-technical users: new graphical user interface, word processing, drawing,
painting, 3D modeling, animation, music composing and editing, information management, hypermedia and multimedia authoring (HyperCard, Director), and network information environments (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 gradually shifted to software environments: graphic design, architecture, product design, space design, filmmaking, animation, media design, music, higher education, and culture management.

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 beginning in the middle of the 1980s, GUI-based software quickly put computer in the center of culture. Theoretically, I think we should think of the subject of software in the most expanded way possible. That is, we need to consider not only “visible” software used by consumers but also “grey” software, which runs all systems and processes in contemporary society. Yet, since I don’t have personal experience writing logistics software or industrial automation 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 and which I would call cultural software. While this term has previously used metaphorically (see J.M. Balkin, *Cultural Software: A Theory of Ideology*, 2003), in this book I am using this term literally to refer to software programs which are used to create and access media objects and environments. The examples are programs such as Word, PowerPoint, Photoshop, Illustrator, Final Cut, After Effects, Flash, Firefox, Internet Explorer, etc. Cultural software, in other words, is a subset of application software which enables creation, publishing, accessing, sharing, and remixing images, moving image sequences, 3D designs, texts, maps, interactive elements, as well as various combinations of these elements such as web sites, 2D designs, motion graphics, video games, commercial and artistic interactive installations, etc. (While originally such
application software was designed to run on the
desktop, today some of the media creation and
editing tools are also available as webware, i.e.,
applications which are accessed via Web such as
Google Docs.)

Given that today the multi-billion global culture
industry is enabled by these software programs, it is
interesting that there is no a single accepted way to
classify them. Wikipedia article on “application
software” includes the categories of “media
development software” and “content access
software.” This is generally useful but not completely
accurate – since today most “content access
software” also includes at least some media editing
functions. QuickTime Player can be used to cut and
paste parts of video; iPhoto allows a number of
photo editing operations, and so on. Conversely, in
most cases “media development” (or “content
creation”) software such as Word or PowerPoint is
the same software commonly used to both develop
and access content. (This co-existence of authoring
and access functions is itself an important
distinguishing feature of software culture). If we are
visit web sites of popular makes 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 commonly gets – another reason
why we should focus our theoretical tools on
interrogating cultural software.

In this book my focus will be on these applications
for media development (or “content creation”) – but
cultural software also includes other types of
programs and IT elements. One important category
is the tools for social communication and sharing of
media, information, and knowledge such as web
browsers, email clients, instant messaging clients,
wikis, social bookmarking, social citation tools,
virtual worlds, and so on- in short, social software
(Note that such use of the term “social software”
partly overlaps with but is not equivalent with the
way this term started to be used during 200s to refer to Web 2.0 platforms such as Wikipedia, Flickr, YouTube, and so on.) Another category is the tools for personal information management such as address books, project management applications, and desktop search engines. (These categories shift over time: for instance, during 2000s the boundary between “personal information” and “public information” has started to dissolve disappeared as people started to routinely place their media on social networking sites and their calendars online. Similarly, Google’s search engine shows you the results both on your local machine and the web – thus conceptually and practically erasing the boundary between “self” and the “world.”) Since creation of interactive media often involves at least some original programming and scripting besides what is possible within media development applications such as Dreamweaver or Flash, the programming environments also can be considered under cultural software. Moreover, the media interfaces themselves – icons, folders, sounds, animations, and user interactions - are also cultural software, since these interface mediate people’s interactions with media and other people. (While the older term Graphical User Interface, or GUI, continues to be widely used, the newer term “media interface” is usually more appropriate since many interfaces today – including interfaces of Windows, MAC OS, game consoles, mobile phones and interactive store or museums displays such as Nanika projects for Nokia and Diesel or installations at Nobel Peace Center in Oslo – use all types of media besides graphics to communicate with the users.8) I will stop here but this list can easily be extended to include additional categories of software as well.

Any definition is likely to delight some people and to annoy others. Therefore, before going forward I would like to meet one likely objection to the way I defined “cultural software.” Of course, 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 behavior 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 media-creating tools. Am I saying that today “culture” is equated with particular subset of application software and the cultural objects 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 in the end of the 20th century humans have added a fundamentally new dimension to their culture. This dimension is software in general, and application software for creating and accessing content in particular.

I feel that the metaphor of a new dimension added to a space is quite appropriate here. 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.” In other words, 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, and so on. So while we can certainly study “the culture of software” – look at things such as 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 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. In other word, just as adding a new dimension of space adds a new coordinate to every element in this space, “adding” software to culture changes the identity of everything which a culture is made from.
In other words, 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 which together make up “culture.”

As just one example of how the use of software reshapes even most basic social and cultural practices and makes us rethink the concepts and theories we developed to describe them, consider the “atom” of cultural creation, transmission, and memory: a “document” (or a “work”), i.e. some content stored in some media. In a software culture, we no longer deal with “documents,” “works,” “messages” or “media” in a 20th century terms. Instead of fixed documents whose contents and meaning could be full determined by examining their structure (which is what the majority of twentieth century theories of culture were doing) 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 browsing a web site, use Gmail, play a video game, or use a GPS-enabled 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. Computer programs can use a variety of components to create these “outputs”: design templates, files stored on a local machine, media pulled out from the databases on the network server, the input from a mouse, touch screen, or another interface component, and other sources. Thus, although some static documents may be involved, the final media experience constructed by software can’t be reduced to any single document stored in some media. In other words, in contrast to paintings, works of literature, music scores, films, or buildings, a critic can’t simply consult a single “file” containing all of work’s content.
“Reading the code” – i.e., examining the listing of a computer program – also would not help us. First, in the case of any real-life interactive media project, the program code will simply be too long and complex to allow a meaningful reading - plus you will have to examine all the code libraries it may use. And if we are dealing with a web application (referred to as “webware”) or a dynamic web site, they often use multitier software architecture where a number of separate software modules interact together (for example, a web client, application server, and a database). (In the case of large-scale commercial dynamic web site such as amazon.com, what the user experiences as a single web page may involve interactions between more than sixty separate software processes.)

Second, even if a program is relatively short and a critic understands exactly what the program is supposed to do by examining the code, this understanding of the logical structure of the program can’t be translated into envisioning the actual user experience. (If it could, the process of extensive testing with the actual users which all software or media company goes through before they release new products – anything from a new software application to a new game – would not be required.) In short, I am suggesting “software studies” should not be confused with “code studies.” And while another approach - comparing computer code to a music score which gets interpreted during the performance (which suggests that music theory can be used to understand software culture) – appears more promising, is also very limited since it can’t address the most fundamental dimension of software-driven media experience – interactivity.

Even in such seemingly simple cases such as viewing a single PDF document or opening an photo in a media player, we are still 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 show will only tell us some things about the experience that we would get when we interact with this document via software. While the content’s of the file obviously forms a part of this experience, it is also shaped by the interface and the tools provided by software. This is why the examination of the assumptions, concepts, and the history of culture software – including the theories of its designers - is essential if we are to make sense of “contemporary culture.”

The shift in the nature of what constitutes a cultural “object” also calls into questions even most well established cultural theories. Consider what has probably been one of the most popular paradigms since the 1950s – “transmission” view of culture developed in Communication Studies. This paradigm describes mass communication (and sometimes culture in general) as a communication process between the authors who create “messages” and audiences that “receive” them. These messages are 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 consider such partial reception a problem; in contrast, from the 1970s Stuart Hall, Dick Hebdige and other critics which later came to be associated with Cultural Studies argued that the same phenomenon is positive – the audiences construct their own meanings from the information they receive. But in both cases theorists implicitly assumed that the message was something complete and definite – regardless of whether it was stored in some media or constructed in “real time” (like in live TV programs). Thus, the audience member would read all of advertising copy, see a whole movie, or listen to the whole song and only after that s/he would interpret it, misinterpret it, assign her own meanings, remix it, and so on.

While this assumption has already been challenged by the introduction of timeshifting technologies and
DVR (digital video recorders), it is just does not apply to “born digital” interactive software media. When a user interacts with a software application that presents cultural content, this content often does not have definite finite boundaries. For instance, a user of Google Earth is likely to find somewhat different information every time she is accessing the application. Google could have updated some of the satellite photographs or added new Street Views; new 3D building models were developed; new layers and new information on already existing layers could have become available. Moreover, at any time a user can load more geospatial data created by others users and companies by either clicking on Add Content in the Places panel, or directly opening a KLM file. Google Earth is an example of a new interactive “document” which does not have its content all predefined. Its content changes and grows over time.

But even in the case of a document that does correspond to a single computer file, which is fully predefined and which does not allow changes (for instance, a read-only PDF file), the user’s experience is still only partly defined by the file’s content. The user is free to navigate the document, choosing both what information to see and the sequence in which she is seeing it. In other words, the “message” which the user “receives” is not just actively “constructed” by her (through a cognitive interpretation) but also actively “managed” (defining what information she is receiving and how.)

**Why the History of Cultural Software Does not Exist**

“Всякое описание мира сильно отстает от его развития.”
(Translation from Russian: “Every description of the world seriously lags behind its actual development.”)
Тая Катюша, VJ on MTV.ru.10
We live in a software culture - that is, a culture where the production, distribution, and reception of most content - and increasingly, experiences - 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, GIMP, Final Cut, After Effects, Blender, Flash, Maya, or MAX.

Where does contemporary cultural software came from? How did its metaphors and techniques were arrived yet? And why was it developed in the first place? We don’t really know. Despite the common statements that digital revolution is at least as important as the invention of a printing press, we are largely ignorant of how the key part of this revolution - i.e., cultural software - was invented. Then you think about this, it is unbelievable. Everybody in the business of culture knows about Guttenberg (printing press), Brunelleschi (perspective), The Lumiere Brothers, Griffith and Eisenstein (cinema), Le Corbusier (modern architecture), Isadora Duncan (modern dance), and Saul Bass (motion graphics). (Well, if you happen not to know one of these names, I am sure that you have other cultural friends who do). And yet, a few people heard about J.C. Liicklider, Ivan Sutherland, Ted Nelson, Douglas Engelbart, Alan Kay, Nicholas Negroponte and their collborators who, between approximately 1960 and 1978, have gradually turned computer into a cultural machine it is today.

Remarkably, history of cultural software does not yet exist. What we have are a few largely biographical books about some of the key individual figures and research labs such as Xerox PARC or Media Lab - but no comprehensive synthesis that would trace the genealogical tree of cultural software. And we also don’t have any detailed studies that would relate the history of cultural software to history of media, media theory, or history of visual culture.
Modern art institutions - museums such as MOMA and 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, Silicon Valley does not 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 cultural software). 

I believe that the major reason has to do with economics. Originally misunderstood and ridiculed, modern art has eventually became a legitimate investment category – in fact, by middle of 2000s, the paintings of a number of twentieth century artists were selling for more than the most famous classical artists. Similarly, Hollywood continues to rip profits from old movies as these continue to be reissued in new formats. What about 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’s AutoCAD, and many other popular cultural applications build up 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 a separate products which can be re-issued today. (In principle, I can imagine 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 nostalgia of adults for the cultural experiences of their teenage years and youth by making these experiences into
new products, it is actually surprising that early software versions were not turned into a market yet. If I used daily MacWrite and MacPaint in the middle of the 1980s, or Photoshop 1.0 and 2.0 in 1990-1993, I think these experiences were as much part of my “cultural genealogy” as the movies and art I saw at the same time. Although I am not necessary advocating creating 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 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 “new media,” “media art,” “internet,” “cyberspace,” “cyberculture” and “code,” we lack not only a conceptual history of media editing software but also systematic investigations of its roles in cultural production. For instance, how did the use of the popular animation and compositing application After Effects has reshaped the language of moving images? How did the adoption of Alias, Maya and other 3D packages by architectural students and young architects in the 1990s has similarly influenced the language of architecture? What about the co-evolution of Web design tools and the aesthetics of web sites – from the bare-bones HTML in 1994 to visually rich Flash-driven sites five years later? You will find frequent mentions and short discussions of these and similar questions in articles and conference discussions, but as far as I know, there have 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 usually are not further developed.

**Summary of the book’s argument and chapters**

Between early 1990s and middle of the 2000s, cultural software has replaced most other media
technologies that emerged in the 19th and 20th century. Most of today's culture is created and accessed via cultural software - and yet, surprisingly, few people know about its history. What was the thinking and motivations of people who between 1960 and late 1970s created concepts and practical techniques which underlie today's cultural software? How does the shift to software-based production methods in the 1990s change our concepts of "media"? How do interfaces and the tools of content development software have reshaped and continue to shape the aesthetics and visual languages we see employed in contemporary design and media? Finally, how does a new category cultural software that emerged in the 2000s – "social software" (or "social media") – redefined the functioning of media and its identity once again? These are the 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 new creative techniques it enables across different 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.

While new media theorists have spend 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 Englebardt, Ted Nelson, Alan Kay, and other pioneers of cultural software working in the 1960s and 1970s – 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 have worked 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.

I suggest that Kay and others 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 new 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 1960s and late 1980s. 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 – Word (1984), PageMaker (1985), Illustrator (1987), Photoshop (1989), After Effects (1993), and others.

So what happens next? Do Kay’s theoretical formulations as articulated in 1977 accurately predict the developments of the next thirty years, or have there been new developments which his concept of “metamedium” did not account for? Today we indeed use variety of previously existing media simulated in software as well as new previously non-existent media. Both are 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?

In part 2 I look at the next stage in the development of media authoring software which historically can be centered on the 1990s. While I don’t discuss all the different mechanisms responsible for the continuous development and expansion of computer metamedium, I do analyze in detail a number of them. What are they? At the first approximation, we can think of these mechanisms as forms of remix. This should not be surprising. In the 1990s, remix has gradually emerged as the dominant aesthetics of the era of globalization, affecting and re-shaping everything from music and cinema to food and fashion. (If Fredric Jameson once referred to post-modernism as “the cultural logic of late capitalism,” we can perhaps call remix the cultural logic of global capitalism.) Given remix’s cultural dominance, we may also expect to find remix logics in cultural software. But if we state this, we are not yet finished. There is still plenty of work that remains to be done. Since we don’t have any detailed theories of remix culture (with the possible exception of the history and uses of remix in music), calling something a "remix" simultaneously requires development of this theory. In other words, if we simply label some cultural phenomenon a remix, this is not by itself an explanation. So what are remix operations that are at work in cultural software? Are they different from remix operations in other cultural areas?

My arguments which are developed in part 2 in the book can be summarized as follows. 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 most 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.

To describe how previously separate media work together in a common software-based environment, I coin a new term “deep remixability.” Although “deep remixability” has a connection with “remix” as it is usually understood, it has its own distinct mechanisms. Software production environment allows designers to remix not only the content of different media, but also their fundamental techniques, working methods, and ways of representation and expression.

Once they were simulated in a computer, previously non-compatible 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 popular Google Earth application that combines techniques of traditional mapping, 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.”

While today “deep remixability” can be found at work in all areas of culture where software is used, I focus on particular areas to demonstrate how it functions in detail. The first area is motion graphics – 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 1990s is directly related to 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.

Why did I select motion graphics as my central case study, as opposed to any other area of contemporary culture which has either been similarly affected by the switch to a software-based production processes, or is native to computers? The examples of the former area sometimes called “going digital” are architecture, graphic design, product design, information design, and music; the examples of the later area (referred to as “born digital”) are game design, interaction design, user experience design, user interface design, web design, and interactive information visualization. Certainly, most of the new design areas which have a word “interaction” or “information” as part of their titles and which emerged since middle of the 1990s have been as ignored by cultural critics as motion graphics, and therefore they demand as much attention.

My reason has to do with the richness of new forms – visual, spatial, and temporal – that developed in motion graphics field since it started to rapidly grow after the introduction of After Effects (1993-). If we approach motion graphics in terms of these forms and techniques (rather than only their content), we will realize that they represent a significant turning point in the history of human communication techniques. Maps, pictograms, hieroglyphs, ideographs, various scripts, alphabet, graphs, projection systems, information graphics, photography, modern language of abstract forms (developed first in European painting and since 1920
adopted in graphic design, product design and architecture), the techniques of 20th century cinematography, 3D computer graphics, and of course, variety of “born digital” visual effects – practically all communication techniques developed by humans until now are routinely get combined in motion graphics projects. Although we may still need to figure out how to fully use this new semiotic metalanguage, the importance of its emergence is hard to overestimate.

I continue discussion of “deep remixability” by looking at another area of media design - visual effects in feature films. Films such as Larry and Andy Wachowski’s Matrix series (1999–2003), Robert Rodriguez’s Sin City (2005), and Zack Snyder’s 300 (2007) are a part of a growing trend to shoot a large portion or the whole film using a “digital backlot” (green screen). These films combine multiple media techniques to create various stylized aesthetics that cannot be reduced to the look of twentieth century live-action cinematography or 3D computer animation. As a case study, I analyze in detail the production methods called Total Capture and Virtual Cinematography. They were originally developed for Matrix films and since then has used in other feature films and video games such as EA SPORT Tiger Woods 2007. These methods combine multiple media techniques in a particularly intricate way, thus providing us one of the most extreme examples of “deep remixability.”

If the development of media authoring software in the 1990s has transformed most professional media and design fields, the developments of 2000s – the move from desktop applications to webware (applications running on the web), social media sites, easy-to-use blogging and media editing tools such as Blogger, iPhoto and iMovie, combined with the continuously increasing speed of processors, the decreasing cost of noteboos, netbooks, and storage, and the addition of full media capabilities to mobile phones – have transformed how ordinary people use media. The exponential explosion of the number of
people who are creating and sharing media content, the mind-boggling numbers of photos and videos they upload, the ease with which these photos and videos move between people, devices, web sites, and blogs, the wider availability of faster networks – all these factors contribute to a whole new “media ecology.” And while its technical, economic, and social dimensions have already been analyzed in substantial details – I am thinking, for instance, of detailed studies of the economics of “long tail” phenomena, discussions of fan cultures, work on web-based social production and collaboration, or the research within a new paradigm of “web science” – its media theoretical and media aesthetics dimensions have not been yet discussed much at the time I am writing this.

Accordingly, Part 3 focuses on the new stage in the history of cultural software - shifting the focus from professional media authoring to the social web and consumer media. The new software categories include
social networking websites (MySpace, Facebook, etc.), media sharing web sites (Flickr, Photobucket, YouTube, Vimeo, etc.); consumer-level software for media organization and light editing (for example, iPhoto); blog editors (Blogger, Wordpress); RSS Readers and personalized home pages (Google Reader, iGoogle, netvibes, etc). (Keep in mind that software – especially webware designed for consumers – continuously evolves, so some of the categories above, their popularity, and the identity of particular applications and web sites may change by the time you are reading this. One graphic example is the shift in the identity of Facebook. Suring 2007, it moved from being yet another social media application competing with MySpace to becoming “social OS” aimed to combine the functionality of previously different applications in one place – replacing, for instance, stand-alone email software for many users.)

This part of the book also offers additional perspective on how to study cultural software in
society. None of the software programs and web sites mentioned in the previous paragraph function in isolation. Instead, they participate in larger ecology which includes search engines, RSS feeds, and other web technologies; inexpensive consumer electronic devices for capturing and accessing media (digital cameras, mobile phones, music players, video players, digital photo frames); and the technologies which enable transfer of media between devices, people, and the web (storage devices, wireless technologies such as Wi-Fi and WiMax, communication standards such as Firewire, USB and 3G). Without this ecology social software would not be possible. Therefore, this whole ecology needs to be taken into account in any discussion of social software, as well as consumer-level content access / media development software designed to work with web-based media sharing sites. And while the particular elements and their relationship in this ecology are likely to change over time – for instance, most media content may eventually be available on the network; communication between devices may similarly become fully transparent; and the very rigid physical separation between people, devices they control, and “non-smart” passive space may become blurred – the very idea of a technological ecology consisting of many interacting parts which include software is not unlikely to go away anytime soon. One example of how the 3rd part of this book begins to use this new perspective is the discussion of “media mobility” – an example of a new concept which can allow to us to talk about the new techno-social ecology as a whole, as opposed to its elements in separation.