Lev Manovich

Cultural Software

From new introduction to Software Takes Command manuscript.

July 2011 version.

Software, or the Engine of Contemporary Societies

Search engines, recommendation systems, mapping applications, blog tools, auction tools, instant messaging clients, and, of course, platforms which allow others to write new software – iOS, Android, Facebook, Windows, Lunix, – 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.[1] 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

[1] This and next paragraphs were written for the (unrealized) proposal for *Software Society* book put together by me and Benjamin Bratton for The MIT Press in 2003.

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.

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 relatively little direct attention. Even today when people are constantly interacting with and updating dozens of apps on their mobile phones and other computer devices, "software" as a distinct 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 Open Source movement and related issues around copyright and IP that has been extensively discussed in many academic disciplines. In the last few years, we also see a rapidly growing number of trade books about Google, Facebook, Amazon, and other web giants. Some of these books offering insigntful discussions of the software and its underlying concepts as developed by these companies and the social, political, cognitive, and epistomological effects of this software (such as Google web search engine).[2] In this respect, journalists are way ahead of academics who are still need to catch up.

If we limit critical discussions of digital culture to the notions of "open access", "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"?

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 The MIT Press in 2001), where, as far as I know, the terms "software studies" and "software theory" appeared for the first time. I wrote:

[2] For an exellent example, John Battelle. The Search: How Google and Its Rivals Rewrote the Rules of Business and Transformed Our Culture. Portfolio Trade, 2006, accessed January 21, 2008.

"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 and cultural, social, and economic forces which are shaping development of software itself.

The book which first comprehensively demonstrated the necesssity of the second appoach 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, *New Media 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 pionnering 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, Fuler wrote: "Software is often a blind spot in the theorisation 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."[3]

I completely agree with Fuller that "all intellectual work is now 'software study." Yet it will take some time before the intellectuals will realise it. To help bring this change, in 2008, Mathew Fuller, Noah Wardrip-Fruin and me established Software Studies book series at MIT Press. The already published books include 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), and Code/Space: Software and Everyday Life by Rob Kitchin and Martin Dodge (2011). (Email any of us if you want to submit your own book proposal in software studies.) And in 2011, Fuller together with a number of UK researchers established Computational Culture, open-access peer-reviewed journal which will provide a platform for more publications and discussions.

[3] http://pzwart.wdka.hro.nl/mdr/Seminars2/softstudworkshop, accessed January 21, 2008.

In the Foreword to Software book series, Fuller writes:

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.[4]

Indeed, a number of earlier works by the leading media theorists of our times - Katherine Hayles, Friedrich A. Kittler, Lawrence Lessig, Manual Castells, Alex Galloway, and others - can also be retroactively identified as belonging to "software studies."[5] 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 other words, the state of "software studies" is similar to where "new media" was in the middle of the 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

[4] Software Studies series introduction. http://mitpress.mit.edu/catalog/browse/browse.asp?btype=6&serid=179, accesed July 14, 2011.

[5] See Michael Truscello. *Behind the Blip: Essays on the Culture of Software (review) Cultural Critique* 63, Spring 2006, pp. 182-187.

that a new academic discpline 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 agenta of existing disciplines and which can be studied using aleady exising methods – for instance, Latour's 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 demonstates 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 the intellectuals who have most systematicaly 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, Wendy Hui Kyong, Mathew Fuller, Alexander Galloway, Ian Bogust, Geet Lovink, Paul D. Miller, Peter Lunenfeld, Katie Salen, Eric Zimmerman, Matthew Kirschenbaum, William J. Mitchell, Bruce Sterling In contrast, the scholars without this technical experience such as Jay Bolter, Siegfried Zielinski, Manual Castells, and Bruno

Latour have not included discusssions of software in their overwise theoretically precise and 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 T*he 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 Processing, PHP, and ActionScript. Another important factor is the publication of APIs by all major Web 2.0 companies in the middle of 2000s. (API, or Application Programming Interface, is a code which 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 made programming iself 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 intergrate all the functionality offered by Google Maps into your site, this is a great motivation for beginning to work with Javascript.

In his 2006 article which reviewed other examples of new technologies which 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.”[6] A few years later, this is exacly what happened. In june 2011, 450,000 aps were available on Apple App Store, and 225,000 aps on Android Market[7].

Inspite of these very impressive numbers, the gap between people who can program and who can’t remains. Clearly, today the consumer technologies for capturing and editing media are much easier to use than even most high level programming and scipting 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 gradualy getting more democratised. 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.

[6] 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>, accessed March 23, 2008.
[7] For the current stats on the number of iOS apps and number of times they were downloaded, see http://en.wikipedia.org/wiki/App_Store_(iOS)#Milestones, accessed July 2, 2011,

What is "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."[8] Kittler himself programms in an assembler language which probably determined his distrust of Graphical User Interfaces and modern software which uses 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 characterised by tools such as assembler languages.

While Software Studies (as already defined by growing number of books and articles) is concerned with all types of software, my own particular interests are with **cultural software**. While this term was used earlier metaphorically (for instance, 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 which support actions we normally associate with "culture." These cultural actions enabled by software can be divided into a number of categories:

[8] Friedrich Kittler, 'Technologies of Writing/Rewriting Technology' <http://www.emory.edu/ALTJNL/Articles/kittler/kit1.htm>, p. 12; quoted in Michael Truscello, "The Birth of Software Studies: Lev Manovich and Digital Materialism," *Film-Philosophy,* Vol. 7 No. 55, December 2003. http://www.film-philosophy.com/vol7-2003/n55truscello.html, acccessed January 21, 2008.

1) creating, sharing and accessing cultural artifacts which contain representations, ideas, beliefs, and aesthetic values (for instance, editing a music video or designing a package for a product);

2) engaging in interactive cultural experiences (for instance, playing a computer game);

3) creating and sharing information and knowledge (for instance, writing an article for Wikipedia, adding places in Google Earth);

4) communicating with other people (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.[9]);

5) participating in online information ecology (for instance, adding to information available to Google web search software for generating future search results everytime you use this service, clicking "+1" button on Google+ or 'Like' button on Facebook.)

6) developing software tools and services which support all these activities (for instance, programming a library for Processing which enables sending and receiving data over internet[10])**.**

Technically, this software may implemented in a variety of ways. Popular implementations (refered in computer industry as "architectures") include stand-alone *applications* which run on user computing device, *distributed applications* (a client running on user device commnicates with softare on

[9] http://en.wikipedia.org/wiki/Facebook_features, accessed July 14, 2011.
[10] http://www.processing.org/reference/libraries/, accessed July 7, 2011.

the server[11]), and *peer-to-peer networks* (each computer becomes both a client and a server[12]). If all this sounds completely unfamiliar, don't worry: all you need to understand is that "cultural software" as I will use this term covers a wide range of products and servers (as opposed to only refering to Word, Photoshop, and Firefox). This, all these qualify as cultural software: professional film and video editing and visual effects applications which need special computer hardward beoynd what a typical laptop offers (i.e., Smoke, Flame and Lustre from Autodesk[13]), consumer app iMovie, social media and social network services such as Facebook and Vimeo. In the latter case, the software includes multiple programs and databases running on company's servers (for instance, Google is thought to have over one million servers around the world[14]) and a website and/or apps used by people to send emals, chat, post updates, upload video, leave comments, etc.

Lets go through some of the software types which I listed above in a little more detail.

The first category is *application software for accessing, creating, distributing, and managing (or" publishing", "sharing", and "remixing") media content.* The examples are Microsoft Word, Powerpoint, Photoshop, Illustrator, After Effects, Firefox, Internet Explorer, and Blogger. This category is in the center of this book. Therefore, to be able to refer to it category via a single simple term, I will call it *media software*.

[11] http://en.wikipedia.org/wiki/Client-server, accessed July 7, 2011.
[12] http://en.wikipedia.org/wiki/Client-server#Comparison_to_peer-to-peer_architecture, accessed July 7, 2011.
[13] http://usa.autodesk.com/flame/, accessed July 7, 2011.
[14] "Pandia Search Engine News – Google: one million servers and counting". Pandia Search Engine News. July 2, 2007. Retrieved February 14, 2010.

I will take for granted that since we all use application programs, or "apps," we all have a basic understanding of this term.[15] Similarly, I also assume that we do understand what "content" refers in digital culture, but just to be sure, here are 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., to both enable and reward users to creating, distribute, and re-use pieces of "content" on multiple scales – 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 web stes and blogs, etc[16]. (This modularity parallels the fundamental principle of modern software engineering to desigxn comple programs from small reusable parts called functions or procedures.) All such parts also quality as "content."

Between late 1970s and middle of 2000s, application programs for media editing were designed to run on a user's computer (micicomputers, PCs, scientific workstation, 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

[15] For a possible taxonomy of types of application software, see Application software (n.d.). In *Wikipedia*. http://en.wikipedia.org/wiki/Application_software, accessed July 2, 2011.
[16] For an extended discussion of modularity in new media, see *The Language of New Media* (MIT Press, 2001).

web sites (Google Docs, Picnik photo editor), while others are intergrated with media hosting or social media sevices (i.e. Photobucket image and video editor). Many applications are implemented as clients which run on mobile phones (i.e, Maps on iPhone), tablets, and TVs televison platforms and communicate with the servers and web sites. The examples of such platforms are Apple's iOS[17], Google's Android[18], and LG's Smart TV App platform[19].

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

Cultural software also includes tools and services which are specifically designed for (or at least include comprehensive tools for) *communication and sharing of information and knowledge*, i.e. "social software.[20] The examples incllude search engines, web browsers, blog editors, email clients and services, instant messaging clients, wikis, social bookmarking, social networks, virtual worlds, Massively Multiplayer Online Games and prediction markets. The familiar names include growing family of Google products (Google Web search, Gmail, Google Maps, etc.), Skype, MediaWiki, and World of Warcraft.

[17] For the current stats on iOS apps, see http://en.wikipedia.org/wiki/App_Store_(iOS)#Milestones, accessed July 7, 2011.
[18] http://en.wikipedia.org/wiki/Android_Market, accessed July 7, 2011.
[19] http://www.wired.com/gadgetlab/2011/01/lg-smart-tv/, accessed July 7, 2011.
[20] See http://en.wikipedia.org/wiki/Social_software, accessed July 2, 2011

Of course, people do not share everything online – at least, not yet. 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 which are included with every computer device being sold.

These categories are always changing gradually over time. For instance, during 2000s the boundary between "personal information" and "public information" has being reconfigured as people started to routinely place their media on media sharing sites, and also communicate with other on social networks.

In fact, the whole reason behind existence of *social media and social networking services and hosting web sites* 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 also insure the continuos 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 this service.)

As many of these services begun to offer more and more advanced media editing and information mangememt tools along with their original media hosting and communication and social networking functons, they did manage to largely erase another set of boudaries (from the PC era): between application programs, operating system, and data. Facebook, in particular, was very agrresive as positioning itself as a complete "social platform" which can replace varius stand alone communication programs and services.

Until the rize of social media and proliferation of mobile media platforms, it was possible to study media production, dissemination and consumption as separate processes. Similarly, we could usually separate between production tools, distributon techologies, and media access devices and platforms – for example, TV studio, cameras, lighting, and editing machines (production), transmission systems, and television sets (access). Social media and cloud computing in general erase these boundaries in many cases (especially user-generated content) 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" (which I myself envoked earlier) while always keeping in mind that the current social media / cloud computing paradigms are systematically reconfure what these terms may refer to.

Finally, I need to add one last set of distinctions to this map of cultural software. I am interested in how *software appears to users* – i.e. what *functions* it offers to create, share, reuse, communicate, manage and organize, *media interfaces* used to present these functions, and *assumptions and models about a user, her needs, and society* encoded in these functions and their presentation.

Functions are embedded in app commands, menu, and the choices they offer – in other words, what you can do with a given app, and how you can do it. However, I do want to make a point about media forms of software. Many people still think that contemporary computer devices use Graphical User Interface (GUI). In reality, original GUI of the early 180s (icons, folders, menus) have been gradually extended to include other media and senses – sounds, animations, and vibration feedback which may accompaning user interactions on a mobile device, voice input, multi-touch gesture interfaces, etc. So this is why the term "media interface" is more accurate description of how interfaces work today.

Why a Comprehensive History of Cultural Software Does Not Exist

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 Flash, Photoshop, GIMP, Final Cut, After Effects, Blender, Flame, Maya, MAX, or Dreamweaver.

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? Certain software platforms, services and tools have been extensively covered in media, so their history is relatively well-known (think of Facebook, Google, and Apple). But this is the tiny part of the tip of the iceburg. Media creating and edting software history is pretty much unknown. 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., media software - was invented. Then you think about this, it is unbelieavable. People 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). (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 heard about J.C. Liicklider, Ivan Sutherland, Ted Nelson, Douglas

Engelbart, Alan Kay, and their colloborators who, between approximately 1960 and 1978, have gradually turned computer into a cultural machine it is today.

Remarkably, history of cultural software as its own 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 Media Lab - but no comprehensive synthesis which would trace the geneological tree of media tools.[21] And we also don't have any detailed studies which 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[22]).

I believe that the major reason has to do with economics. Originally misunderstood and ridiculed, modern art has eventualy became a legitimate

[21] The two best books on the pioneeres of cultural computing, in my view, are Howard Rheingold, *Tools for Thought: The History and Future of Mind-Expanding Technology* (The MIT Press; 2 Rev Sub edition, 2000), and M. Mitchell Waldrop, *The Dream Machine: J.C.R. Licklider and the Revolution That Made Computing Personal* (Viking Adult, 2001).
[22] For the museum presentation on the web, see http://www.computerhistory.org/about/, accessed March 24, 2008.

investment category – in fact, by middle of 2000s, the paintings of a number of twentiteh 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 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 technlogies used in these original versions – but, in contast 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 surpising 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 genelogy" 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 "social media," "social networks," "new media," media art," "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 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 arhitectural students and young architects in the 1990s, and the tools available in these programs in different periods have similarly influenced the langauge 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 talks, 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.

How to "Understand Media"?

In Introduction to his *Exressive Processing* published in The MIT Press Software Studies series, Noah Wardrip-Fruin summarizes this sitiation: "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."[23] Wardrip-Fruin and others already started filling this gap – especially in the areas of game platforms and design, and electronic literature.

[23] Noah Wardrip-Fruin. *Expressive Processing: Digital Fictions, Computer Games, and Software Studies*. The MIT Press, 2011.

In this respect, the related fields of code studies and platform studies being developed by Mark Marino[24], Nick Montfort and Ian Bogost are playing a very important role. According to Marino (and I completely agree), these three fields compliment each other in this way: "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)."[25]

Despite significant progress in the last few years, much work will remains. In particular, *the processes of media design using application software* have not been analyzed in detail. This is exactly what my book aims to address.

The book has three related goals.

The first goal is to better **understand the media objects** which we experience and engage with hundreds of times every day: animated TV titles, TV and web animated ads, graphic designs, illustrations, web graphics and banners, and so on. Very often, these artifacts are parts of interactive media experiences – navigating the web, playing a video games The examples of "engagment" are sharing, editing, remixing, and commenting. (So-called "social media block" buttons which are standard today on lots of web sites exemplify these forms of engagments.)

[24] Mark C. Marino, "Critical Code Studies." *Electronic Book Review*, December 12, 2006. http://www.electronicbookreview.com/thread/electropoetics/codology, accessed July 14, 2011.
[25] http://chnm2011.thatcamp.org/05/24/session-proposal-critical-code-studies/, accessed July 14, 2011.

This media is experienced, created, edited, remixed, organized and shared with software. The software includes stand-alone professional media design applications such as Photoshop, Dreamwether, After Effects, Aperture, Illustrator, Maya, and Word; consumer-level apps such as iPhoto, iMovie, or Picassa; and social media tools (editing / sharing / commenting) provided by social media sites such as Facebook, Vimeo, and Photobucket. Therefore, my second goal is to **understand media software** – its geneology (where does it come from), its anatomy (the key features shared by all media viewing and editing software), and its effects in the world. Specifically, I will be concerned with two kinds of effects: 1) How media design software shapes the media being created, making some design choices seem natural and easy to execute, while hiding other design possibilities; 2) How media viewing / managing / remixing software shapes our experience of media and the actions we perform on it.

My third goal is to **undestand what "media" is** today conceptually. Are the concepts of media developed to account for industrial era technoogies, from photography to video still work in relation to media which is designed and experienced with software? Do they need to be updated, or completely replaced by new more appopriate concepts? For example: do we still have different media or did they merged into a single new meta-medium? Are there some structural features which motion graphics, graphic designs, web sites, product designs, buildings, and video games all share since they are all designed with software?

In short: does "media" still exist?