Lev Manovich

SOFTWARE STUDIES

[2008]

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. Lets 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]

---

[1] http://money.cnn.com, accessed January 21, 2008.

[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 which 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 which also daily appears 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 gllobal 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"?

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 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, 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 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. At the moment of this writing (Spring 2008), software studies is a new paradigm for intellectual inquiry which is now just beginning to emerge. The very first book which has this term in its title is being published by The MIT Press later this year (*Software Studies: A Lexicon*, edited by Matthew Fuller.) At the same time, a number of

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

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.[4] 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 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

---

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

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, 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 overwise 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 their 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."[5] A few years later, the apps exploded,

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.

---

[5] 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.

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 shap by culture in its turn. The time for "software studies" has arrived.

Why the History of Cultural Software Does not Exist

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."[6] 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.

Although Software Studies is concerned with all software, we have a particular interest in what I call *cultural software*. While this term has

[6] 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.

previously used metaphorically (for instance, see J.M. Balkin, *Cultural Software: A Theory of Ideology*, 2003), in this book I am using this term literally to refer to programs such as Word, Powerpoint, Photoshop, Illustrator, After Effects, Firefox, Internet Explorer, and so on. Cultural software, in other words, is a particular subset of *application software aimed at creating, distributing, and accessing (or publishing, sharing, and remixing) cultural objects* such as images, movies, moving image sequences, 3D designs, texts, maps, as well as various combinations of these and other media. (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.)

Cultural software also includes *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, virtual worlds, and so on.[7] I also would also include under cultural software *tools for personal information management* such as address books, project management applocations, and desktop search engines. (These categories themselves are not absolute but are shifting over time: for instance, during 2000s the boundary between "personal information" and "public information" has increasingly dissappeared as people started to routinely place their their media on social networking sites; similarly, Google's search engine shows you the results both on your local machine and the web.) Last but not least, *the media interfaces* themselves – icons, folders, sounds and animations accompaning user interactions - are

[7] See http://en.wikipedia.org/wiki/Social_software, accessed January 21, 2008.

also cultural software, since these interface mediate people's interactions with media and other people.

We live in a software culture - that is, a culture where the production, distribution, and reception of most content is mediated by software. And yet, most creative professionals do not know anything about the intellectual history of software they use daily - be it Photoshop, 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 unbelieavable. 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 colloborators 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 a few laregly 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 cultural software.[8] 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[9]).

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

---

[8] 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).

[9] For the museum presentation on the web, see http://www.computerhistory.org/about/, accessed March 24, 2008.

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 "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 arhitectural students and young architects in the 1990s has 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 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.