Beyond Engineering

Software Design as Bridge over the Culture/Technology Dichotomy

Bernhard Rieder and Mirko Tobias Schäfer

Abstract In this chapter, we first consider the growing cultural significance of software as a motive for having a closer look at software production. We then show how networked computing has stimulated new practices of technical creation that question the traditional logic of engineering; *open source* software development serves as an example. Consequently, it is no longer feasible to separate the technological dimension from its cultural context. An integrated perspective could lead both humanities scholars and technologists to revaluate established dichotomies and refocus the debate on technological policies.

1 Introduction

In his book "*Le Geste et la Parole*", the paleontologist André Leroi-Gourhan sketched the evolution of Homo sapiens as leaving the domain of biological advancement to continue, with an accelerated pace, in the field of language and technology. While many of Leroi-Gourhan's proposals have not aged well, his concept of humanity being shaped by a man-made web of objects and symbols – of machinery and discourse one might say – has been a powerful image in a time when the idea of the tool as neutral artifact is still an important paradigm. In the last decade there has been a resurgence of academic interest in technology, not purely as a means to an end but as a cultural force. Together with this shift in perspective on the role of technical artifacts in our high-tech collectives, we see, more specifically, an increased awareness of the "toolmaker" as the assumed *locus* of technical progress. Every age seems to have an epitomical figure of technical creation: the craftsman for the Middle Ages, the inventor in the Industrial Revolution, and the engineer in the 20th century. Late capitalism has introduced a new figure for the beginning of the 21st century: the *designer* as the toolmaker of the information age.

B. Rieder, Paris 8 University

M. T. Schäfer, Utrecht University

The last two decades have produced a plethora of literature on the new mode of creating technical objects: from product design to Web design, from industrial design to experience design, design is everywhere but no two definitions are the same. As a consequence, the term refers less to a clear-cut concept or methodology; rather it functions as a means of differentiation. Software design[1] for example is not a well-defined practice: it is a way of saying that what is being done is somehow going beyond the well-defined practice of software engineering. Behind the term "design" actually lurks a multiplicity of quite different ways of creating, shaping, and maybe even using.

2 Hybrid Practices

In industrial societies there remain few tasks that are not in one way or another dependent on computers. Our communication and information routines have shifted in a large part to a computer-based network infrastructure of globally connected computers, the *metamedia* (Kay and Goldberg, 1977) of our time. Classic electronic media like television and telephony are currently passing onto the universal protocol of TCP/IP,[2] becoming yet another piece of software that runs on the Internet. Creative work, game play, social intercourse, information search and management, so many of the things we do in our everyday lives have become directly connected to digital tools and networks (Castells, 2000). We are steering towards a unified digital environment in which computer hardware and software define possibilities for action and conditions of expression.

Interest in technology within the humanities has historically been limited. When considered, technical artifacts have been assimilated into the industrial complex and treated as producers of *capital* rather than of *meaning*. But the dense entanglement between human and non-human we witness today increasingly calls for perspectives that zoom in at the micro-level and theorize not only the general aspects of how "society and culture" relate to "technology," but first and foremost the increasingly hybrid everyday practices that are the content of human affairs.

In reference to de Certeau (1980), we can describe these practices as ways of doing that embed actions in a dense network of meaning, provide a rationale for why something is done, and sketch a proper way of doing it. There is a non-discursive dimension to such an *art de faire*, e.g., motor movement, objects, and spatial settings, and a strong discursive element, e.g., morals, laws, rules, and narratives. These two aspects are woven together by continuous action. Collins and Kusch (1998) have detailed how the atomic particles of practices, actions, can themselves be theorized as series or trees of micro-acts, coalescing motor movement

[1] The term was first coined in Kapor (1986).

[2] Transmission Control Protocol / Internet Protocol are the communication protocols that unite all the different networks that make up the Internet.

and meaning. And Actor-Network-Theory has shown (Latour, 1999) that actions are not properties of individual agents, but of chains linking human and non-human "actants", combining each ones "program of action" to form hybrid actors. If we understand practice as an embedding of action in time and habit, in these views, the discursive dimension of an *art de faire* cannot be severed from its non-discursive, mechanic counterpart.

When applying this view, we see that in general, and with ICT in accelerated and enlarged form, machines are responsible for always larger parts of the action trees or action chains, rendering actions intrinsically hybrid. As a consequence, our practices have become riddled with the work of machines, in many cases without us even noticing. Software – the prime interest of this chapter – now goes even deeper than "classic" technology because many of the tasks being delegated to *logical* machinery are *semantic* in nature. Among other things, algorithms now filter, structure, interpret, and visualize information in an automatic fashion, performing tasks previously reserved for humans.

From a practical standpoint, we can understand this process of hybridization along two axes: new actions and practices are becoming possible, e.g., drawing on a virtual canvas, video communication across oceans, and real-time data-mining, and existing actions and practices are done in new ways, e.g., different in form, style, speed, efficiency, difficulty, and range.

In this sense, software is responsible for extending, both quantitatively and qualitatively, the role that technology plays in the everyday practices that make up modern life. Culture and technology are intertwined at the micro-level, to the extent that even the analytical separation of the two becomes highly problematic (Latour, 1999). Is separation between a discursive and a non-discursive level still possible when computer programs analyze email, news bulletins, and scientific publications to decide which ones to bring to our attention and which ones silently to discard? When the visibility of an opinion becomes a question of algorithms,[3] meaning is deeply embedded in the non-discursive: in the software itself. Technology is not only surrounded by discourse, it is discourse. Although we do not share Heidegger's hostile stance toward technology, his understanding of the tool as an ontological agent, as a way of "Entbergen" (revealing), is still worth considering. In "Gestell" (enframing), the discursive and the non-discursive conflate; it is both object and logic – a *diagram*, in the terms of Foucault, but with the difference in nature between the two planes largely gone. The lesson we take from this is diametrically opposed to Heidegger's position: involvement instead of withdrawal.

We would like to argue that technology affords not one but multiple ways of revealing being, and that the way we create technical artifacts – and software most importantly – heavily influences the cultural role they will play. Tools are not neutral; they integrate and propagate human values (Friedman, 1997). But these

[3] The Slashdot communication platform (http://www.slashdot.org) for example uses an elaborate discussion system that includes a technological measure of symbolic capital and modulates the visibility of individual messages accordingly.

values are not necessarily those of technocratic reasoning as Heidegger would have it, the whole gamut of human apprehension is possible. Software brings technology closer to us than ever before and it is time to look at the practices that spawn what has become an important part of the constitutional fabric of our cultures.

3 Software, Design and Open Source

Since the advent of modern computing in the late forties and especially the marketing of the consumer PC in the eighties, computers have come to be ubiquitous. But while the terms "computer" and "technology" have almost become synonymous and the basic technical principles have remained the same for the last sixty years, there remains an aura of vagueness around these machines. Herein actually lays their power. Computers themselves are functionally underdetermined; they need software to turn them into complete devices with distinct functions. While the hardware, the *Universal Machine*, coupled with peripherals like input/output devices, networks, etc., is the necessary mechanical base layer, the "specific" machine – a series of functions and procedures that manipulate information and, with proper connection, matter and energy – is the result of programming. Alan Turing stated that,

> The importance of the universal machine is clear. We do not need to have an infinity of different machines in doing different jobs. A single one will suffice. The engineering problem of producing various machines for various jobs is replaced by the office work of 'programming' the universal machine to do these jobs. (1984, 4)

These words mark the technical novelty and yet another reason for the cultural significance of IT: somebody who buys a computer today gets not only the physical apparatus, but also gains access to a seemingly infinite world of logical machinery. These software programs spring from a burgeoning environment where work styles nowadays go well beyond the classical methods of engineering or even beyond the "office work" mentioned by Turing. Before we can get a closer look at these practices, we must first review some of the qualities of software.

3.1 Properties of Software

While there has been a continuous reflection of what software actually is, this problem is still far from being completely understood. Despite the stability of the mathematical foundations of software since Turing, Church, and Shannon, the final jury on what we can really do with it is still out. As society changes, software changes and every day there are new applications that surface around the globe. It is possible, however, to specify some of the basic properties of *logical machinery*.

Unlike other technological objects, software is immaterial. It is similar to language with respect to structure and similar to technology with respect to effect. Written as a text, it functions like a machine. Latour (1992) pointedly observes, paraphrasing Austin, that "how to do things with words and then turn words into things is now clear to any programmer." The classical distinction made in engineering between *designing*, i.e., drawing the blueprints, and *building*, i.e., assembling the physical structure, does therefore not translate well into software programming. According to Jack W. Reeves (1992) writing the source code can be compared to designing but building is nothing but the automatic translation of source code into machine language by a compiler program. In contrast to classic (hardware) engineering, software is thus expensive to design – it takes a lot of time to write a functional piece of software– but cheap to build. From an economic perspective, we can even speak of an apparatus of production unlike other areas of technology, specific to the creation of software: except for the price of a computer, producing software is basically free, time becoming the essential cost factor. In this sense, software is again closer to literature or music than to industrial production – the workstation is the factory floor. This greatly facilitates people shifting from consumers to producers.

Like knowledge and information, software can be shared without tangible loss for the giver. The Internet transports and copies computer code as simply as text, sound, or images; algorithms, program libraries, and modules pile up at different sites, contributing to what could be seen as the equivalent of a fully equipped workshop with an unlimited spare parts inventory attached to it, accessible again at the cost only of time and skill. A general-purpose programming language like Java nowadays comes with thousands of ready-made building blocks and writing code is often closer to playing Lego than to the laborious task of manipulating memory registers it used to be.

Unlike the products of industry, a computer program is always tentative, never really finished or "closed". Classic machinery also has to be tended, calibrated, and repaired, but with software the provisional aspect is pushed to the extreme. One mouse click and an entire subsystem can be copied to another program and the output of one piece of software can instantly become the input of another. We do not want to encourage in any way the view that holds that everything digital is fluid, chaotic, and auto-organized, but there remains the fact that this freedom from most physical constraints renders software easier to manipulate and handle than hardware objects. The only constraining factors are time and skill. This relative freedom is one reason for the production of software in practice being so unlike engineering by the book.

3.2 Software Design as Heterogeneous Practice

According to IEEE Standard 610.12, software engineering is "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."[4] The attempt to translate the strategies and methods of

[4] See: http://standards.ieee.org/catalog/olis/se.html

classic engineering into the area of software has never been entirely successful and has been criticized from several directions. We cannot possibly summarize all the different views expressed in this complex and long-standing debate, but there are several main critical positions that can be distinguished.

One argument holds simply that programming is based less on method than on skill, that it is craftsmanship rather than engineering, and that "in spite of the rise of Microsoft and other giant producers, software remains in a large part a craft industry" (Dyson, 1998). The main question for design, then, is not how to find the proper methods but how to acquire the appropriate skills.

Another argument is that software engineering has its place but that specific methods and strategies cannot be directly imported from traditional engineering, because building software is very different from building bridges and houses (Reeves, 1992). Debugging for example should not be treated as a hassle to be eliminated by using mathematical rigor, but as an essential part of creating computer programs.

Finally there are those who believe that software engineers should be supplemented by other professions, in particular by software designers who take inspiration from architects rather than engineers because buildings and software "stand with a foot in two worlds – the world of technology and the world of people and human purposes" (Kapor, 1996). In this view, building a computer program is not so much about technical problems, but about how to bring users and tools together in a meaningful way.

Independent of these different views the empiric observation remains that the practice of creating software rarely resembles the top-down engineering models like the *lifecycle*- or the *waterfall*-model where the process of going from neat requirements to a working program is thought of as an advancing in clear cut stages. The "real world" of software development is most often described as "messy, ad hoc, atheoretical" (Coyne, 1995), as consisting of "bricolage, heuristics, serendipity, and make-do" (Ciborra, 2004), or as the result of "methodological and theoretical anarchism" (Monarch et al., 1997). While this does not automatically make software production "art", as Paul Graham (2003) suggests, we have to accept that the engineering ideal is just that: an ideal. Software production in practice commonly takes paths that go in different ways beyond engineering. Two important factors have to be taken into account: changing problems and increasing complexity.

First, the problems software is expected to be used to solve are becoming more "cultural" and less "technical." If computers were still doing what they did in the 1960s, namely number crunching and data storage, there would probably be no discussion about software engineering or design. With computers now performing semantic and social functions this has changed. Methods like *participatory design* or *end-user development* are now used to try to integrate the fuzziness of specifications for software by integrating future users into the construction process.

Second, the complexity of software is increasing rapidly and this makes it always more difficult to plan a program in every detail before starting to write code. It is often impossible to foresee problems early on and plans and models have to be changed, tests have to be made, and specifications have to be modified during the construction process. Agile methods like *extreme programming* and *rapid-prototyping* strive to make complexity more manageable and transform the top-down waterfall into a long series of iterations.

The properties of software, the distribution of these properties into space by means of the Internet, and the changing technological landscape are slowly eroding the modern ideal of a neat separation between technology and culture, between detached rationality and human motivations. This argument is endorsed by a closer look at the diverse landscape of software production. As an example, we will briefly analyze the *open source* scene to show how a whole new array of actors, strategies, and practices can emerge in a situation where material cost is no longer a limiting factor.

3.3 The Open Source Scene

On one level, the term "open source" refers to a certain way of handling and sharing computer software.[5] It implies that programs are not just available in machine code, but also in source code, i.e., in text files written in a programming language accessible to human beings. To qualify as open source, it is essential that the public is allowed to modify and redistribute the product. On another level, the term refers to communities[6] built around this notion of openness and sharing that is responsible for a considerable amount of today's software production. There is now an *open source* equivalent for nearly every type of program

The *open source* scene is rather diverse, but it is possible to sketch a rough ideal type for how it functions. Most importantly, it is impossible to imagine open source without the existence of the Internet. Platforms like sourceforge.net, along with mailing lists and newsgroups, are the tools used to organize and coordinate a globally dispersed and mostly voluntary workforce. A project usually starts with an embryonic program written by an individual or a group which is released under an *open source* license, to people who are invited to participate in its development. If it can stimulate enough interest, a lively process is set in motion: following the "release early, release often" maxim, versions of the program are regularly published on the Web where anybody interested can add code, report bugs, and fix them. Which features and fixes are integrated is usually decided by a moderator (group or individual), supplemented by a community process very similar to scientific peer-review. The very linear structure of classic engineering is thus translated into a rapid succession of coding/building/debugging, where requirements specification, interface design, and user testing are carried out concurrently and

[5] We are referring here to the open source definition given by the Open Source Initiative (http://www.opensource.org/docs/definition.php).

[6] The *open source* scene is far from homogenous and there is some infighting between the very political *Free Software Movement* and the rather pragmatic *Open Source Movement*.

subject to constant change. Collaboration is the main "tool" to tackle complexity. The Internet-based development platforms provide the infrastructure for a project's representation, for communication between its participants and for the coordination of bug tracking and code maintenance; they are the media that render possible what could be called a "virtual factory" where a diverse and dispersed public channels its collective intelligence.

The *open source* scene also distinguishes itself from traditional engineering in social norms and general mindset. Mathematical rigor is valued less than an open and involved communication style. Similar to other (youth) subcultures, the demonstration of skill (and not diplomas) is the main source of symbolic capital. Inclusiveness, discussion, collaboration, and the open circulation of information is more important than the clear-cut attribution of tasks, positions, and responsibilities.

On an institutional level, the *open source* scene has become an important element in the socialization and education of programmers. The lively and helpful online communities allow one to get help and learn from individuals who have achieved status based on their contribution to the field. The accessible code landscape and participatory culture of the open source scene make for a powerful learning environment for individuals of all levels of skill. While engineering is traditionally connected to the somewhat authoritarian institutions of school and university, the open source community supplements these forms by offering a learning-by-doing environment based on playful imitation and autodidactic skill acquisition.

To show that *open source* products are an important part of the software landscape, we will briefly discuss three examples: the *Linux* operating system, the *Apache* Web server, and the Internet browser *Firefox*.

Linux started out in 1991 when a Finnish student, Linus Torvalds, wrote a very basic kernel program, the core of any operating system, as a hobby project and released it on the Web, inviting others to participate. Since then, *Linux* has developed into a modern, robust, and complete operating system and is now probably the only serious competitor for Microsoft Windows left. It is available for free and constantly maintained and extended by a community of thousands of programmers around the globe. Most Fortune 500 companies now use *Linux*, as do the metropolitan administrations of Vienna, Munich, and Paris. One reason for this success is cost, but other factors come into play, including reliability, platform independence, and the possibility to fix bugs directly without having to go through a vendor company.

The *Apache* project was initiated in 1995 and has since then steadily grown to become the dominant Web server application with a market share of over 52%.[7] *Open source* and available for free, it is developed and maintained under the guidance of the *Apache Software Foundation*, a non-profit company that helps to organize the development process, assures legal support for the community, and protects the brand. *Linux* and *Apache*, coupled with the free database system

[7] Netcraft ServerWatch July 2007, http://www.serverwatch.com/stats/article.php/3686926

MySQL and an *open source* programming language, *PHP*, form the most common platform (called LAMP) for dynamic Web applications.

The *Firefox* Web browser grew out of code released to the community in 1998 by the ailing company Netscape. After several rather unsuccessful products, the *Mozilla Foundation* released *Firefox* at the end of 2004 as version 1.0. Carried by strong critique of Microsoft's *Internet Explorer* for its various security leaks, the open source browser captured considerable market share[8] in 2005. It is also a good example for how the open source community allows for the participation of non programmers. Using *Bugzilla*, a tool for tracking bugs, anybody can report errors and ask for features in future releases. Skilled users may extend the browser through plug-ins without having to get to know the code of the main application. *Firefox* is finally not just a piece of software, it is also a community providing logos, T-shirts, images, and wallpapers as well as an entire viral marketing campaign.

The *open source* scene shows that methods and strategies in technical production cannot be divorced from the social, economic, and cultural environment they are stimulating and being stimulated by. The *culture of engineering* is but one of many possibilities in a field that has opened up to manifold models for production. Computers have made technical creativity accessible to a larger and more diverse audience than any previous technologies have. From writing code to designing levels for computer games, there is a wide scale of possible involvement for every level of skill. While the new modes of creation are in many ways similar to earlier forms of amateur culture they are different in a very important aspect: the three programs we discussed are not just niche products but highly competitive artifacts of great quality that hold strong market positions. This signals an *extended culture industry*, where the production of cultural artifacts opens up to the formerly excluded: the consumers.[9] There are of course many commercial actors playing a role in the *open source* scene – IBM, Novell, Intel, and others take an active part in financing and developing. However, the intertwined networks of production that span companies and individuals go beyond the mono-directional processes Adorno and Horkheimer (1944) have criticized so severely. The idea has been contagious and phenomena like *Wikipedia*, blogging, or the countless music labels on the Web take the *open source* principle to a larger context of cultural production. Computers and the Internet can be seen as enabling technologies that give users the opportunity to extend the culture industry and to participate in the production of cultural artifacts, stimulating the social dynamic we are witnessing today (Jenkins, 2002)-recently branded around the term "Web 2.0".

While engineering is often seen as a neutral, detached, and "objective" way of problem-solving, the collaborative and auto-organized design process that marks

[8] In Europe Firefox is ranging up to 34% in Finland and 24% in Germany; see XiTi Browser Survey, September 2005, online: http://www.xitimonitor.com/etudes/equipement11.asp

[9] According to Walter Benjamin (2002), facilitating the transformation from consumers to producers is every artist's political obligation.

the *open source* scene does not strive to separate the social and cultural aspects of technological creation from the task of designing and writing code.

These developments are not necessarily aimed at replacing the traditional, and more organized institutions of work, education, and research; what we witness today is a trend toward plurality and cross-fertilization. With reference to Eric Raymond (1998), we could say that the bazaar does not supplant the cathedral but blossoms in the city streets around it, slowly infiltrating the sacred halls; and the development of "alternative" methods and strategies for the production of software is by no means limited to the open source community: because of the increasing complexity and "culturalization" of computing problems mentioned above, most fields are constantly forced to go beyond established methodology. Taken together, we see software design as a shifting field that unites a plurality of heterogeneous methods, mindsets, and actors.

4 Bridging the Culture/Technology Divide

So far, we have made two separate arguments: first, we have tried to show that software plays an increasingly important role in our everyday lives, accentuating culture as a hybrid of technology and discourse. Second, we have discussed how software production flourishes outside of the classical institutions and methodology of engineering. In the third part of this chapter, we want to briefly discuss these two arguments in relation to their impact in three different areas: the humanities, technology, and policymaking.

4.1 The Humanities Discourse

Traditionally philosophy and cultural theory have subscribed to a view of technology as something external to, or at least different from, society and culture. In this perspective, the practice of creating a technical artifact is very dissimilar in nature from processes of symbolization, e.g., the writing of law or literature. The first is supposedly oriented toward the material domination of our "lifeworld" (Lebenswelt) through efficiency, while the second is concerned with the social (law) or cultural (literature) dimension of human existence. This separation has the convenient effect of exempting those thinking about technology to have any need for technical knowledge because "techno-science" always produces only more of the same, the true challenge lying in the discovery of the essential dynamics between the strata, an endeavor reserved to the masters of symbolization. However, there is a very dangerous side to this outlook: subtracting the dimension of *meaning* from technology implies the subtraction of *responsibility*. If the creation of technology is not understood to be a deeply cultural, social, symbolic, and political activity, there is no reason for the creators to adopt any ethical and political stance toward their work beyond the question of physical harm to others. We believe that in a time when the use of logical machinery is a part of so many of the practices that make up our lives, we need concepts that take into account not only the "effects" of technology on culture, but which recognize that technology *is* a form of culture: embodying not just the homogenous logic of "Gestell," but being continuously differentiated into a plurality of forms, practices, values, and power struggles.

There is a growing amount of empirical work on large software projects to which social scientists have contributed. However, looking at the field of software design we should ask whether our concepts of technology are adequate for grasping the multiplicity of possible connections between methodologies, the artifacts they produce, and the consequences for society. The humanities could take up the task of broadening our still very restrained technological imagination and lead the way towards modes of production that facilitate finding other liaisons between the human and non-human than those marked only by domination, efficiency, and convenience.

4.2 The Technologist Discourse

If we recognize software design as a pluralistic and fractured practice which takes a part in shaping the fabric of the world in which we live, we have to rethink our stance not only as theorists, but also as creators of technology. Terry Winograd and Fernando Flores wrote nearly twenty years ago that "we encounter the deep question of design when we recognize that in designing tools we are designing ways of being" (Winograd and Flores, 1986). A dialogue between the different groups implicated in designing software is necessary to foster awareness of the cultural dimension of their work. A start has already been made: a part of the *open source* community has adopted an explicit stance on the political issues surrounding their technical efforts and the software design community is making a strong effort to link up with the humanities.

The field that is lagging severely behind is education. There is still very little discourse between technical departments and the humanities, and current curricula are neither fit for producing the "culturally-aware technologist" nor the "technically-aware theorist". Herein lies the true challenge of bridging the dichotomy between culture and technology: bringing the more inclusive understanding of technology that is currently emerging to places where it can have an effect.

4.3 Policies

The third area of our discussion is policy, and luckily there is already a very lively debate going on in this area, especially around the questions of software patents and *open source*. The discussion however is strongly centered on economic and juridical questions, treating the cultural aspects as mere collaterals. It is rarely recognized that the creators of technology, operating outside of the classic pathways of established industry, are a crucial part of civil society in that they actively produce means for expression and action. Only when we understand writing software as one possible way of participating as a citizen can the political issues be properly addressed. The state, as the arbiter in the ongoing battle around software patents, will have to decide whether the amorphous coder communities sprawling on the Web, that put their work at the disposition of the public domain, are of special value to society and therefore worth protecting against the overwhelming financial capacities of the established commercial actors. The new design practices that we have tried to present and theorize in this chapter are by no means inevitable; although the Universal Machine is a strong base for the social and cultural activities surrounding them, the free flourishing of technical creativity is a fragile thing that can easily be reduced to the point of mere hobbyist dabbling, as it was the case with many other technologies. There is (still) democratic potential in the new metamedia and we will have to decide whether we want to nurture it or not.

5 Conclusion

We have entitled this chapter "beyond engineering", because the term "engineering" has come to stand for the technocratic separation between a sphere of technology and a sphere of culture, society, and politics; for a mindset that treats the creation of technical artifacts as a detached and orderly process, closer to calculation than to creativity. The modern ideal of engineering as a politically and culturally neutral process – unspoiled by human motivations and uncontaminated by morals and emotions – appears today to be rather anachronistic. A closer look at software *design* shows that there are multiple methods, strategies, and mindsets guiding the creation of programs, systems, and applications. Our short analysis of the *open source* scene is evidence that extensions to classic methodologies, alternative routes, collaborative approaches, and auto-organized forms of workflow are both possible and effective.

We believe that the fluctuations in how technical artifacts are created are not just minor adjustments but necessary adaptations to the changing place of technology in our societies. As technology infiltrates the practices that make up our everyday lives, culture *stabs back* by invading the terrain of production, bringing all its contingencies, contradictions, and complexities along. Their separation was never clear anyhow, but the level of interpenetration has reached new heights today. The immaterial qualities of software, distributed into space using the global infrastructure of the Internet, affect an increasing number of people, users as well as designers. We have called the resulting space of production, distribution, and consumption an *extended culture industry* where the boundaries between consumers and producers are blurring and social and technical forces are closely intertwining.

While there is some understanding of how to channel social forces in a democratic fashion, it is still unclear how we can achieve the same for the technical part of the hybrid. It now seems evident that in high-tech societies the creation of tools and objects plays an important role in shaping cultural practice, expression, and imagination; it is a highly cultural gesture. Looking at the similarities between language and software can help us to understand the nature of our currently complicated techno-social situation; it can also make us see that freedom of technical creation is a form of freedom of speech. It is the duty of the humanities to seek out what that could mean.

References

Adorno, T., and Horkheimer, M., 1988, *Dialektik der Aufklärung*, Fischer, Frankfurt a. M., first published in 1944.

Benjamin, W., 2002, Der Autor als Produzent, in: W. Benjamin, *Medienästhetische Schriften*, Suhrkamp, Frankfurt a. M., pp. 231–247, first published in 1934.

Castells, M., 2000, *The Information Age: Economy, Society and Culture*, Blackwell, Malden, MA, 3 volumes, first published in 1996.

Certeau, M. de, 1994, *L'invention du quotidien*, Gallimard, Paris, first published in 1980.

Ciborra, C., 2004, Encountering information systems as a phenomenon, in: *The Social Study of Information and Communication Technology: Innovation, Actors, and Contexts*, C. Avgerou, C. Ciborra, and F. Land, Oxford University Press, Oxford, pp. 17–37, p. 19.

Collins, H., and Kusch, M., 1998, *The Shape of Actions: What Humans and Machines Can Do*, MIT Press, Cambridge, MA.

Coyne, R., 1995, *Designing Information Technology in the Postmodern Age: From Method to Metaphor*, MIT Press, Cambridge, MA, p. 32.

Dyson, F. J., 1998, Science as a craft industry, *Science* 280(5366):1014–1015.

Friedman, B., ed., 1997, *Human Values and the Design of Computer Technology*, Cambridge University Press, Cambridge.

Graham, P., 2003, Hackers and Painters, Lecture at Harvard, http://www.paulgraham.com/hp.html

Jenkins, H., 2002, Interactive audiences?, in: *The New Media Book*, D. Harries, ed., British Film Institute, London, pp. 157–170.

Kay, A., and Goldberg, A., 2003, Personal dynamic media, in: *The New Media Reader*, F. Wardrip and N. Montford, eds., MIT Press, Cambridge, MA, pp. 393–404, first published in 1977.

Kapor, M., 1996, A software design manifesto, in *Bringing Design to Software*, T. Winograd, ed., Addison-Wesely, Boston, pp. 1–10, p. 4.

Latour, B., 1992, Where are the missing masses?, in: *Shaping Technology / Building Society*, W. Bijker and J. Law, eds., MIT Press, Cambridge, MA, pp. 225–258, p.255.

Latour, B., 1999, *Pandora's Hope: Essays on the Reality of Science Studies*, Harvard University Press, Cambridge, MA.

Monarch, I. A., Konda, S. L., Levy, S. N., Reich, Y., Subrahmanian, E., and Ulrich, C., 1997, Mapping sociotechnical networks in the making, in: *Social Science, Technical Systems, and Cooperative Work: Beyond the Great Divide*, G. C. Bowker, S. L. Star, W. Turner, and L. Gasser, eds., Lawrence Erlbaum Associates, Mahwah, pp. 331–354, p. 337.

Raymond, E. S., 1998, The cathedral and the bazaar, *First Monday* **3**(3), http://www.firstmonday.org/issues/issue3_3/raymond/

Reeves, J. W., 1992, What is software design?, *C++ Journal*, Fall 1992.

Turing, A. M., 1948, *Intelligent Machinery*, National Physical Laboratory Report (http://www.alanturing.net/turing_archive/archive/l/l32/L32-001.html).

Winograd, T., and Flores, F., 1986, *Understanding Computers and Cognition: A New Foundation for Design*, Addison-Wesley, Boston, p. xi.