Christoph Neubert

'The Tail on the Hardware Dog'. Historical Articulations of Computing Machinery, Software, and Services

2015

https://doi.org/10.25969/mediarep/1019

Veröffentlichungsversion / published version
Sammelbandbeitrag / collection article

Empfohlene Zitierung / Suggested Citation:
Neubert, Christoph: 'The Tail on the Hardware Dog'. Historical Articulations of Computing Machinery, Software, and Services. In: Irina Kaldrack, Martina Leeker (Hg.): *There is no software, there are just services.* Lüneburg: meson press 2015, S. 21–37. DOI: https://doi.org/10.25969/mediarep/1019.

“The Tail on the Hardware Dog”: Historical Articulations of Computing Machinery, Software, and Services

Christoph Neubert

The emergence of service-oriented business models in the computer industry over the last 15 years is part of broader historical dynamics underlying the relations between hardware, software, and services. This article traces the changing configurations of this triad with a particular focus on the economic, technological, and social construction of “software” in exemplary contexts. The historical evidence opens analytical and critical perspectives on the current rearticulation of software in terms of “services.”

There is no software. It strikes one as a historical paradox that this claim, defended by Friedrich Kittler in the early 1990s with critical rigor against the ideology of human control over seemingly transparent computer hardware (Kittler 1992, 1993, 2014), resonates with business models hailed by today's computer industry under the labels of *Software as a Service* (SaaS) and *Service-oriented Architecture* (SOA). Taking this paradox seriously, I will consider the idea of an epochal transition from software to services pursued by the present volume under a broader historical perspective, starting with the observation that the distinction between hardware, software, and services does not lie in the nature of things, but is a product of complex historical processes. In essential respects, the current convergence of software and services reverses a historical development: The proposition that there is no software but only services describes a situation characteristic of the computer industry until the 1970s. The supposed decline of software has thus to be evaluated in the light of the emergence and transformation of "software" as technical artifact, economic good, and social dispositive. Witnessing its disappearance, the question arises: How did software come into being in the first place?

Systems and Programs

In the context of computing, the first usage of the term "software" in print is ascribed to the statistics professor John W. Tukey in 1958 (Shapiro 2000). The word was probably coined earlier verbally and in working papers, perhaps by Paul Niquette in the 1950s (Niquette 2006), or already in the late 1940s by the RAND mathematician Merrill Flood (Cerruzi 2003, 365, 372). However, according to the *Oxford English Dictionary*, the word "software" came into broader use not before the early 1960s, referring to the "body of system programs, including compilers and library routines, required for the operation of a particular computer and often provided by the manufacturer, as opposed to program material provided by a user for a specific task" (OED).

[Fig. 1] "Tin Canned Software" (Cybermatics 1971).

As this description already suggests, the historical notion of software crucially differs from our present understanding in several respects (cf. Haigh 2012). In a narrower sense, the concept comprised systems software such as operating systems, assembly systems, programming tools and compilers. In a wider sense, software was taken to include media such as punched cards and magnetic tapes, but also written documentation and even human activities such as system analysis or training. Being linked closely to computer hardware on the one side, and to all sorts of services on the other, software did not cover what we take as its essence today, namely applications. A second aspect distinguishing the historical from the present understanding is that software was not originally a commercial product: *Operating systems* (OS), utilities, and programming tools were provided free of charge by the hardware manufacturers, being considered part of general services a firm bought or rented together with a hardware installation. Programs for specific business tasks such as payroll, file systems, or accounting, on the other hand,

were highly customized and usually written in-house by the data processing staff of the firms.

For a long time, software represented "only the tail on the hardware dog" (Bender 1968, 243). Accordingly, the software industry emerging since the mid-1960s was marginal and provided programming services rather than standardized products. Even where programs were offered as "canned" solutions (Figure 1), the proposed deal included hardware infrastructure, training, and customization. First attempts to acquire programs that had been developed by individual firms and sell them on a license-basis as packaged applications to other customers in the same business were not undertaken before the late 1960s, and with little success (Brown 2002; Head 2002). Even providing a catalogue of useful software solutions did not meet the customers' needs or expectations (Welke 2002). The idea to pay for software, especially for standardized products that were not even adapted to a firm's specific requirements, seemed to make no sense. The often cited "software crisis"of the 1960s manifested in scarcity of qualified personnel (Ensmenger 2010, 51ff.), but questions of structured product design and the Taylorization of coding labor in an emerging software industry did not become relevant before the 1970s. Indeed, the term *software engineering* in the sense of "the professional development, production, and management of system software" (OED) was first used in 1968 (Mahoney 2004).

Time Sharing

The bias towards services characteristic of the computing industry of the 1950s and 1960s was largely due to enormous hardware costs. Large mainframe and minicomputers represented expensive infrastructures that were supplied to customers in terms of a "computer utility rhetoric." Just as electricity consumers did not keep their own power plants, "it would be cheaper and more reliable for organizations to buy information processing from a service provider, rather than owning a

mainframe computer" (Campbell-Kelly and Garcia-Swartz 2007, 752). The technology underlying this service model is known as *time sharing*. The concept of time sharing was developed in the late 1950s, mainly motivated by the aim to make efficient use of expensive mainframe computers by avoiding idle times. Time sharing refers to the (seemingly) simultaneous access of multiple users that are connected via terminals to a central computer, technically based on the flexible allocation of CPU-time to concurrent user processes. The first experimental implementation, the *Compatible Time Sharing System* (CTSS), was deployed at the MIT in 1961 on an IBM 709 computer, followed in 1963 by the CTSS II on an IBM 7094 that allowed access of 30 remote users (Auerbach 1973, 65). Further time sharing systems were developed in the following years for various platforms by IBM, by Bolt, Beranek, and Newman, and by General Electrics in cooperation with Dartmouth College.

Evolved in universities and research centers, the technology of time sharing translated readily into a business model. The first commercial provider, Adams Associates, appeared in 1963, followed by IBM in 1964 (Auerbach 1973, 65). Even with the advent of IBM's System/360 in the same year, computing hardware remained expensive and installations time and resource consuming, so only larger administrations and firms could afford to rent or even buy the respective equipment and keep the required personnel. Many smaller companies outsourced their data processing activities and took recourse to the services of time sharing providers, who offered remote access over public or private data lines to computing infrastructure including hardware, programming environments (e.g. for COBOL, FORTRAN, and BASIC), software packages, file storage, and print services. Customers typically rented the required terminal equipment and were charged for parameters such as CPU-time, connection time, and storage volume.

Unbundling

The emancipation of a dedicated software industry from the previous economy of hardware and services involved two major steps. The first step was the emergence of the enterprise software sector since the 1970s, which was accompanied by a variety of technological, economic, and social innovations, including the standardization of products, new business and marketing models, a changing mentality of customers, professionalization of programmers, the rise of software engineering and corresponding methods such as structured programming and the systematic reuse of code in terms of software libraries, the development of interpreters and compilers for high-level computer languages, and the introduction of affordable and compatible hardware systems such as the IBM S/360 series (cf. Johnson 2002; Goetz 2002a, 2002b).

The efforts it took to invent software as an economic good and product in its own right is impressively illustrated by the incidents that led IBM to give up the practice of bundling programs with hardware and services. On January 17, 1969, the U.S. Department of Justice filed a suit against IBM, charging the company with monopolizing the general-purpose computer market; the bundling of services, software and machinery was taken to be anti-competitive and illegal. It took the Antitrust Division six years to bring the case to trial in 1975, and it lasted another six years before it was finally dropped in 1982, then considered to have been "without merit" (cf. Johnson 1982; Kirchner 1982). However, in preparation of one of the longest and costliest antitrust trials in history, some 30 billion pages of paperwork were provided. During the trial,

> Some 2,500 depositions were taken in all, and IBM compiled and stored in special warehouses 66 million pages of evidence. At the lawsuit's peak, more than 200 IBM lawyers were working on the case, on whom the company spent tens of millions of dollars annually. [...] The parties called

> 974 witnesses [...] and produced 104,400 pages of testimony. (Anthes 1989, 65)

While the case was negotiated, IBM issued internal directives suppressing the description of programs as products:

> We should realize that discussing [applications] programs separate from the machines in advertising or presentations is inconsistent with our fundamental position that hardware and software including programs are an indivisible product [...]. (cited in Arnst 1977, 4)

On the other hand, IBM reacted very fast to the legal issues raised by the Justice Department. An "unbundling task force" had already been formed in 1966 in the context of introducing the S/360 series (cf. Grad 2002; Humphrey 2002), and on June 23, 1969, IBM announced its decision to pursue separate pricing of hardware, software, and services. This date has been taken as the birth of the software industry or "Independence Day for software firms" (Gibson 1989, 6), though in retrospect, it is more likely that IBM's decision was not the cause but rather a symptom or effect of an emerging business sector. In any case, the unbundling affair provides a striking impression of the complexities raised by the emancipation of software.

Mass Markets and Pricing Models

After the quarrels in the enterprise sector during the 1960s and 1970s, the second major step towards software as a product is linked to the growing impact of the *personal computer* (PC) since the 1980s, which opened a mass market for consumer software (cf. Campbell-Kelly 2001). The PC served as host for packaged software applications offered to customers in *shrink-wrapped* boxes, and this software in turn played an important role for the domestication of computer hardware, its integration into the environments of offices and private households. At the same time, the concepts of *layered architecture* and *protocol stacks*

as formulated in the OSI-reference model allowed to establish basic standards for the interconnection of computers, initiating the transition of the terminal-mainframe logic of enterprise computing towards the client-server logic of intranets and the Internet. Networked computing and the WWW opened new software markets for client and server operating systems (Novell Netware, Microsoft NT), web browsers (Netscape, Microsoft), web publishing software (Macromedia, Adobe), and antivirus software (Symantec).

The Internet business soon blurred the distinction between the economic sectors of services, enterprise software, and consumer software in the reverse order of their historical appearance: mass-market vendors such as Microsoft entered the enterprise software business, and later both consumer and enterprise software vendors turned to services (cf. Campbell-Kelly and Garcia-Swartz 2007, 736f.). The Internet also enabled new forms of collaborative work on programs leading to the *Open Source* movement. The impact of Open Source and "free" software, in particular of Linux, together with other trends such as the rise of mobile media and gadgets, the crash of the Internet economy around 2000, and the increasing commoditization of hardware and proprietary software led to the decline of the software product paradigm established in the 1980s and to new strategies of value generation. One interesting development in this context is the *appliances* model that returns to the idea of bundling proprietary software and hardware as a boxed product (Hein 2007). Appliances in this sense include consumer products such as game consoles, mp3-players, navigation devices, and personal gadgets of all sorts, but also enterprise appliances such as routers, or dedicated equipment for e-mail and firewall services. Another strategy employs marketing platforms for non-software products such as music downloads or e-books, the streaming of multimedia content, the promotion of social and business services, or the bundling of "free" software with advertising. These changes indicate a general turn of the computer industry

from vertical to horizontal integration and an orientation towards downstream revenues and services. The remaining software vendors were accordingly driven towards new pricing models:

> Traditional product sales and license fees have declined, and product company revenues have shifted to services such as annual maintenance payments that entitle users to patches, minor upgrades, and often technical support. (Cusumano 2008, 20).

Besides payment for maintenance, the classic one-time up-front license fee has been replaced by subscription or pay-per-use models that ensure a constant revenue stream, even during economic downturns. Such pricing models have far reaching consequences for the planning, versioning, and maintenance of products (Olsen 2006). In particular, since the development of new software releases and upgrades is mainly motivated by marketing requirements "creating the illusion of a new product to justify the repeated resale of what is fundamentally the same good" (268), the subscription model eliminates the disruptive effects of release cycles. On the other hand, software subscription tends to generate a lock-in of customers, which is problematic especially for small firms and freelancers (see Leistert 2013 on the example of current policies adopted by Adobe).

Architecture of the Cloud

Quite different from promoting subscription under the guise of a "service" is the idea to provide the functionality of software applications in terms of web services: Instead of deploying a copy of software to be installed and run on the customer's site, the vendor hosts the software on his own servers and provides access via the Internet. This business model is highly dependent on technical factors such as network and server performance and thus leads to the more recent paradigm of *cloud computing*. According to the definition provided by the U.S. Department of Commerce's National Institute of Standards and Technology

(NIST), cloud computing comprises three levels of services (Mell and Grance 2011): *Infrastructure as a Service* (IaaS) refers to the provision of computing resources (processing, storage, networks) that can be configured like on-site hardware and used by the customer to "run arbitrary software, which can include operating systems and applications" (3). The underlying virtual machinery is in turn running on a distributed cloud infrastructure with pooled resources. The model *Platform as a Service* (PaaS) refers to virtual development environments that already include operating systems together with "programming languages, libraries, services, and tools supported by the provider" (2f.). SaaS, finally, represents the highest integration level of cloud computing. The customer here uses the functionality of services without managing any infrastructure on the levels of operating systems, development environments, or application software (cf. Gajbhiye and Shrivastva 2014; Crago and Walters 2015).

Technically, the implementation of SaaS conforms to the framework of Service-oriented Architecture (Laplante, Zhang and Voas 2008). SOA extends the logic of *object-oriented programming* to commercial services, turning from algorithms and control structures to software components that are defined in terms of specific properties, functions, and interfaces; these components shall interact without central control in the context of distributed software systems. A web shop, for example, may invoke a number of services offered by different vendors, including database management, payment services, and logistical services, each in turn drawing on a number of subordinate services such as processing web forms, recommendation systems, or tracking options. These components are only loosely coupled, i.e. during an individual process, services are invoked on demand, their discovery, selection and binding being accomplished "on the fly" in a non-predictable way (cf. Turner, Budgen, and Brereton 2003; Gold et al. 2004). Activities in this context are no longer conceived as traditional programming; central process metaphors

instead refer to aesthetic practices in the domains of music and dance—"composition," "choreography," and "orchestration."

In economic terms, SOA and SaaS neatly integrate with the management of business processes. The composition of services is accomplished by specific software tools such as the *Business Process Modeling Language* (BPML), an XML-based standard which is supposed to provide an efficient translation between economic and computational workflow. BPML was later succeeded by the *Business Process Execution Language for Web Services* (WS-BPEL), developed mainly by IBM and Microsoft and elevated to an industry standard by the OASIS consortium (Organization for the Advancement of Structured Information Standards) (cf. Turner, Budgen, and Brereton 2003; Candan et al. 2009). In this context, software has not only ceased to be a product, it also no longer represents a tool employed to accomplish specific business tasks: rather, both domains seem to converge in fulfillment of the old cybernetic dream that business itself becomes a matter of pure programming (i.e. music and dance).

Hidden Environments

The historical sketch provided so far might contribute to our understanding of the current service orientation in several ways: First of all, it becomes evident that the boundaries between hardware, software, and services, as well as the relations between the three domains, are fluid and subject to permanent historical change in conceptual, technological, and economic terms. Second, while hardware on the one hand and services on the other, fit into the classical definition of economic goods and represent fairly stable concepts, the status of software has always been problematic. Since its value depends on configuration, customization, maintenance, and training, software remains closely coupled to services. The emancipation of the shrink-wrapped box seems to represent a transitional phase, and even in the consumer market, complex and costly applications are replaced

today by cheap apps that in many cases function as interfaces to remote services. Third, in economic terms, there is no clear-cut distinction between products and services, which rather represent the endpoints of a continuum. Different business models may rely on different strategies to "servitize" products or to "productize" services (Cusumano 2008, 26). Taken together, there seem to be no simple linear trends, but circular or other dynamics that govern the relations between hardware, software, and services, on micro- as well as on macroeconomic levels (cf. Cusumano 2003, 2008; Suarez, Cusumano, and Kahl 2013). Thus, the present boom of software and computer infrastructure as services can be regarded as a renaissance of essential aspects of the hardware and services computing economy of the 1950s and 1960s.

After all, there is no software. Kittler's speculative and hyperbolic dictum, formulated in the heyday of packaged bit boxes, was obviously inspired by personal experience with personal computers running Microsoft operating systems. But it was meant more generally, pointing to an inevitable strategic delusion rendering invisible the politics and power relations inscribed in hardware. Today, hardware and software retreat from the focus of "user experience" and are supposed to become part of the environment—the "cloud" as a kind of encompassing atmospheric metaphor, or smaller spheres such as the city, the home, clothes, or the human body. Before Mark Weiser formulated the agenda of *Ubiquitous Computing*, the late Marshall McLuhan emphasized the environmental logic of media, drawing on the example of the motor car. McLuhan claimed that the medium is not the vehicle, but the infrastructure, which he further described as a "hidden environment of services" (McLuhan 2005, 242). Thus from the beginning, the concept of "service" links the economy to an ecology of media—a managed ecology, however, of the cybernetic type, which is tuned towards operational closure and blackboxing. In particular, while software promises flexible control over hardware in terms of algorithms, services stand

for the possibility of flexible control over algorithms in terms of functions. While software encapsulates hardware, services encapsulate both hard- and software. In the era of services, both hardware and software are running in *protected mode*.

Coding Services

So what are the real political and ecological conditions of infrastructures? What are the material and energetic resources of the cloud, how are they managed, where, and by whom? How are working conditions in software industries transformed by the service paradigm? As a case in point, we might consider methods such as *Extreme Programming* (XP) or *Agile Programming* (AP) (Beck 1999; Beck et al. 2001) that are historically and systematically linked to SOA and cloud computing (Guha and Al-Dabass 2010; Baliyan and Kumar 2014). Following the requirement of high responsiveness to changing demands, traditional development and production cycles are given up in favor of a general acceleration of workflow. The "agile" paradigm departs from central principles of structured programming and the factory model of software production, considering thorough planning and extensive documentation as harmful. The "Agile Manifesto" and related commentaries (Beck et al. 2001) read as a peculiar combination of working methods with moral values, yielding a work ethic tuned towards efficiency, productivity, and customer satisfaction. While emphasizing categories such as "individuality," "freedom," and "respect," many of the recommended principles and methods are in fact reminiscent of the theory of "egoless programming" formulated in the late 1960s by Gerald Weinberg (Weinberg 1971, 47ff.; cf. Ensmenger 2010, 212–217).

For example, in smaller projects, all team members should be present in the same room, maintain permanent communication, and practice self-monitoring and mutual correction, which is encouraged especially by pair programming in XP. Tasks and roles are flexibly assigned and supposed to change, team

members are brought into direct contact with customers in order to react immediately to their feedback. Programmers are not rewarded for individual skills and competences, but for personal involvement. Work is accomplished by the team as a collective subject. Hierarchies are as flat as possible, central control should be avoided. Thus, in many respects, agile and related programs amount to a convergence of coding technologies and technologies of the self (cf. Neubert 2016). And obviously, the economic ideas of choreography, object-oriented programming, neat cycles, binding on the fly, and flexible work flow, return on the level of programming practices. Like other parts of the service infrastructure, human programmers belong to a pool of resources that are disposable and responsive on demand. In agile methods, the cloud becomes self-referential. Not by coincidence, *Human Capital Management* (HCM) is one of the most profitable services. While structured programming was linked to a Taylorization of software engineering (Mahoney 2004), "agile" programming and related approaches represent a next step towards neo-liberal, perhaps even post-liberal methods of coding subjects.

After all, there surely is a lot of software. So we might have to adjust Kittler's heuristics: *There are no services*.

Bibliography

Anthes, Gary H. 1989. "Rearview Mirror." *Computerworld,* March 2: 63–65.

Arnst, Catherine. 1977. "Bundled Pricing Illegal, 1968 IBM Memo Admits." *Computerworld* 11 (48), November 28: 1, 4.

Auerbach. 1973. *Auerbach Guide to Time Sharing*. Philadelphia, PA: Auerbach Publishers.

Baliyan, Niyati, and Sandeep Kumar. 2014: "Towards Software Engineering Paradigm for Software as a Service." *IC3, 2014 Seventh International Conference on Contemporary Computing (IC3)*: 329–333.

Beck, Kent. 1999. *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley.

Beck, Kent, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon

Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. "Manifesto for Agile Software Development." *agilemanifesto.org*. Accessed April 1, 2015. http://agilemanifesto.org/.

Bender, David. 1968. "Computer Programs: Should They Be Patentable?" *Columbia Law Review* 68 (2): 241–259.

Brown, Walter. 2002. "Founding Atlantic Software." *IEEE Annals of the History of Computing* 24 (1): 80–82.

Campbell-Kelly, Martin. 2001. "Not Only Microsoft: The Maturing of the Personal Computer Software Industry, 1982–1995." *The Business History Review* 75 (1) (*Computers and Communications Networks*): 103–145.

Campbell-Kelly, Martin, and Daniel D. Garcia-Swartz. 2007. "From Products to Services: The Software Industry in the Internet Era." *The Business History Review* 81 (4): 735–764.

Candan, K. Selcuk, Wen-Syan Li, Thomas Phan, and Minqi Zhou. 2009. "Frontiers in Information and Software as Services." *IEEE 29th International Conference on Data Engineering (ICDE)*: 1761–1768.

Ceruzzi, Paul E. 2003. *A History of Modern Computing*. 2nd Edition. Cambridge, MA; London: The MIT Press.

Crago, Stephen P., and John Paul Walters. 2015. "Heterogeneous Cloud Computing: The Way Forward." *Computer* 48 (1): 59–61.

Cusumano, Michael A. 2003. "Finding Your Balance in the Products and Services Debate." *Communications of the ACM* 46 (3): 15–17.

Cusumano, Michael A. 2008. "The Changing Software Business: Moving from Products to Services." *Computer* 41 (1): 20–27.

Cybermatics. 1971. "Tin Canned Software." Advertising. © Cybermatics Inc. *Computerworld* 5 (46), November 17, 1971: 39.

Ensmenger, Nathan. 2010. *The Computer Boys Take Over. Computers, Programmers, and the Politics of Technical Expertise*. Cambridge, MA; London: MIT Press.

Gajbhiye, Amit, and Krishna M. Shrivastva. 2014. "Cloud computing: Need, Enabling Technology, Architecture, Advantages and Challenges." *Confluence. The Next Generation Information Technology Summit. 5th International Conference 25-26 Sept. 2014*: 1–7.

Gibson, Stanley. 1989. "Software industry born with IBM's unbundling." *Computerworld* 23 (25), June 19: 6.

Goetz, Martin. 2002a. "Memoirs of a Software Pioneer: Part 1." *IEEE Annals of the History of Computing 24 (1)*: 43–56.

Goetz, Martin. 2002b. "Memoirs of a Software Pioneer: Part 2." *IEEE Annals of the History of Computing 24 (4)*: 14–31.

Gold, Nicolas, Andrew Mohan, Claire Knight, and Malcolm Munro. 2004. "Understanding Service-Oriented Software." *Software, IEEE* 21 (2): 71–77.

Grad, Burton. 2002. "A Personal Recollection: IBM's Unbundling of Software and Services." *IEEE Annals of the History of Computing* 24 (1): 64–71.

Guha, Radha, and David Al-Dabass. 2010. "Impact of Web 2.0 and Cloud Computing Platform on Software Engineering." *International Symposium on Electronic System Design ISDE 2010*: 213–218.

Haigh, Thomas. 2002. "Software in the 1960s as Concept, Service, and Product." *IEEE Annals of the History of Computing* 24 (1): 5–13.

Head, Robert V. 2002. "The travails of Software Resources." *IEEE Annals of the History of Computing* 24 (1): 82–85.

Hein, Bettina. 2007. *0+0=1: The Appliance Model of Selling Software Bundled with Hardware*. Master Thesis, Massachusetts Institute of Technology.

Humphrey, Watts S. 2002. "Software Unbundling: A Personal Perspective." *IEEE Annals of the History of Computing* 24 (1): 59–63.

Johnson, Bob. 1982. "Justice Department Decides IBM Case 'Without Merit'." *Computerworld* 26 (3), January 18: 1, 8.

Johnson, Luanne. 2002. "Creating the Software Industry: Recollections of Software Company Founders of the 1960s." *IEEE Annals of the History of Computing* 24 (1): 14–42.

Kirchner, Jake. 1982. "Bigness not Bad, Baxter Explains." *Computerworld* 26 (3), January 18: 1, 8.

Kittler, Friedrich. 1992. "There is no Software." *Stanford Literature Review* 9 (1): 81–90.

Kittler, Friedrich. 1993. "Es gibt keine Software." In *Writing/écriture/Schrift*, edited by Hans Ulrich Gumbrecht. München: Fink.

Kittler, Friedrich. 2014."Protected Mode." In Kittler, *The Truth of the Technological World: Essays on the Genealogy of Presence*. Translated by Erik Butler, 209–218. Stanford, CA: Stanford University Press.

Laplante, Phillip A., Jia Zhang, and Jeffrey Voas. 2008. "What's in a Name? Distinguishing between SaaS and SOA." *IT Professional* 10 (3): 46–50.

Leistert, Oliver. 2013. "Mietmodell Software Adobe." *Pop. Kultur & Kritik* 3: 39–42.

McLuhan, Marshall. 2005 [1974]. "Living at the Speed of Light." In *Marshall McLuhan. Understanding Me. Lectures and Interviews*, edited by Stephanie McLuhan and David Staines, 225–243. Cambridge, MA: MIT Press.

Mahoney, Michael Sean. 2004. "Finding a History for Software Engineering." *IEEE Annals of the History of Computing* 26 (1): 8–19.

Mell, Peter, and Timothy Grance. 2011. *The NIST Definition of Cloud Computing: Recommendations of the National Institute of Standards and Technology*. U.S. Department of Commerce. NIST Special Publication 800–145.

Neubert, Christoph. 2016. "Software/Architektur. Zum Design digitaler Dienstbarkeit." In *Dienstbarkeitsarchitekturen. Vom Service-Korridor zur Ambient Intelligence*, edited by Markus Krajewski. Tübingen: Wasmuth. (forthcoming)

Niquette, Paul. 2006. "Softword: Provenance for the Word Software." *niquette.com*. Accessed April 1, 2015. http://www.niquette.com/books/softword/tocsoft.html.

OED. S.v. "software, n." *Oxford English Dictionary Onlin*e. http://www.oed.com/view/Entry/183938.

Olsen, Eric R. 2006. "Transitioning to Software as a Service: Realigning Software Engineering Practices with the New Business Model." *Service Operations and Logistics, and Informatics. SOLI '06. IEEE International Conference, 21-23 June 2006*: 266–271.

Shapiro, Fred A. 2000. "Origin of the Term Software: Evidence from the JSTOR Electronic Journal Archive." *IEEE Annals of the History of Computing* 22 (2): 69–71.

Suarez, Fernando F., Michael A. Cusumano, and Steven J. Kahl. 2013. "Services and the Business Models of Product Firms: An Empirical Analysis of the Software Industry." *Management Science* 59 (2): 420–435.

Turner, Mark, David Budgen, and Pearl Brereton. 2003. "Turning Software into a Service" *Computer* 36 (10): 38–44.

Weinberg, Gerald. 1971. *The Psychology of Computer Programming*. New York, NY: Van Nostrand Reinhold.

Welke, Lawrence. 2002. "Founding the ICP Directories." *IEEE Annals of the History of Computing* 24 (1): 85–89.