Moritz Hiller

Signs o‘ the Times. The Software of Philology and a Philology of Software

2015

https://doi.org/10.25969/mediarep/767

Veröffentlichungsversion / published version
Zeitschriftenartikel / journal article

Signs o' the Times

The Software of Philology and a Philology of Software

Moritz Hiller

Abstract

This paper addresses the question of software preservation by approaching this field from a philologic perspective. Philology here is not understood as hermeneutic operation of interpretation, but rather as practice of preserving material objects: critically providing them as basis for future investigation. Software's status as a material object could not be more uncertain, since it merges – as a source code – a textual dimension and – as a programme – a processual dimension. It is only within the logic of this operativity that software as an object of digital materiality becomes fully conceivable. Since a philology of software would have to consider the phenomenon's dual mode of existence as static text and/or time-critical process to enable research within both dimensions, old questions about what to preserve and how to preserve it rise anew. The paper will therefore take up a few basic notions of traditional scholarly editing, the software of philology. It explores to what extent they can be applied to objects of digital materiality in order to outline an initial idea of a philology of software.

Introduction

Signs o' the times: hidden and invisible, therefore even more effective – but nothing less material. In this day and age of implemented universal machines it is particularly one phenomenon that defines large parts of what is today called culture: software. Thinking about software inevitably leads to a number of difficulties, simply because its status as an object could not be more 'cloudy'. Software merges – as a source code – a textual and – as a programme – a processual dimension. Whereas the mediality of a conventional text is materialised completely in the very moment of its implementation as manuscript, book or digital edition, software's functionality is not fulfilled until it is executed as a programme with/in the hardware of a machine. It is only within the logic of this operativity – which in turn requires physical implementation – that software becomes fully conceivable. Objecting any (still) widespread notion of the immateriality of signs in the realm of digital computer technology, software therein reveals its very material condition.

DOI 10.14361/dcs-2015-0110

For the past ten years, software's multiple modes of existence as a text and/or a process have been the centre of attention of software studies and critical code studies. They allowed for valuable insights into software's materiality, its semiotic and poetic dimensions, as well as its dependency of, entanglement with and power over social, economical and political contexts.[1]

In regard to the question of software preservation – which arises due to software's importance as a cultural artefact and quickly turns into a challenge due to its complex ontological status – the textual dimension of a source code might suggest approaching the task from a textual studies perspective. Cognizant of the shortcomings that the notion of 'text' might produce in this context (cf. Cayley 2002), what is suggested here is not a hermeneutic endeavour to interpret source codes like literary texts, but rather a materialistic practice of a class of objects that are also textual. Scholarly editing has a longstanding tradition of preserving and transmitting such material objects in the sense of research that critically provides the basis for future literary studies.

Fundamentally different from traditional philology, a philology of software would have to consider the phenomenon's dual mode of existence as static text and/or time-critical process, and needs to enable research within both dimensions. Philology has to account for software's operativity: requiring the material implementation and time-critical execution in a machine, that unlike a book is said to be reading and writing itself. Hence, old questions rise anew: What would be an adequate representation of a textuality that transcends traditional notions of text by incorporating materiality and the parameter of time in different ways: simulation or emulation? How do these mimetic operations compare to traditional philologic representations? And to what degree could such a philology rely on hermeneutic premises? Prior to such questions though, the first and foremost task of any philology is to answer the question of its object, which, in the case of software, is a question of digital materiality: What is the 'text' of software then? The following article will therefore take up certain notions of traditional philology, the software of philology, in order to explore how they may be applied to software and to draft an initial outline of a philology of software.

Software

Let us take a step back and begin with the obvious, in the words of the author's own layman understanding: Software, in a lot of cases, is produced by human beings as a means to an end. They write something on a computer that may look like this or at least similar:

```
#include <stdio.h>
int main(void)
{
printf("The familiar form of mostly alphabetic signs is interspersed\n");
```

1 For an overview of the disciplines see Fuller (2008) and Marino (2006).

```
printf("with a few punctuation marks and different special\n");
printf("characters, arranged in-line and line by line.\n");
printf("This already suggests (at least to any reader with\n");
printf("Western literacy) that such a phenomenon relates\n");
printf("to a prevalent form, in spite of any possible semantic oddness:\n");
printf("due to its shape and structure it is most likely to be perceived\n");
printf("as text.\n");
return 0;
}
```

The familiar form of mostly alphabetic signs is interspersed with a few punctuation marks and different special characters, arranged in-line and line by line. This already suggests (at least to any reader with Western literacy) that such a phenomenon relates to a prevalent form, in spite of any possible semantic oddness: due to its shape and structure it is most likely to be perceived as text. The technical term used in German-speaking computer science seems to confirm this presumption. What we see here, a source code, is referred to as *Quelltext*. Does it therefore make sense to assume that the 'text' of software is its source code?

The term 'source code' denotes the textual mode of representation of a computer programme. Humans can generally read it like an alphabetic text, because it is written in the alphanumeric code and syntax of a certain mid or high level programming language (cf. Hagen 2006). In order to be readable for a computer, such a text has to be transformed into the machine code of a particular processor by means of another programme, a so-called compiler (or interpreter).[2] The text printed above is thus not a programme. Actually, in this form – and materiality – it is not even machine executable in principle. Only after the act of transformation into machine code it becomes possible that the text as a programme, as a (not just) literal prescript, can instruct a machine what to do. This again points to the multiple 'states' and layers of the phenomenon of software, which call for a differentiated terminology.

Software studies therefore distinguish between 'delegated code' to "refer to the textual and social practices of source code writing, testing and distribution", focusing on the phenomenon's mode of existence "as a textual source code instantiated in particular modular, atomic, computer-programming languages" (Berry 2011: 31). On the other hand, software studies identify 'prescriptive code' to "include commercial products and proprietary applications, such as operating systems, applications or fixed products of code" (ibid.: 32) – in other words: software. One would be mistaken to think of the two as separate entities though: 'delegated' and 'prescriptive code' are two facets of the same object, embodying the phenomenon's dual character as text and/or process: "code is the static textual form of software, and software is the processual operating form"

2 Disregarded here, in order to not exceed the length of these remarks, is the fact that the object file(s) generated by a compiler still need to be combined by a linker in order to become an executable binary.

(ibid.).[3] Consequently, in the words of Wendy Chun, one could ask: What is *Microsoft Word*? Or, to be more exact: When is *Microsoft Word* actually *Microsoft Word*?

"Is it the encrypted executable, residing on a CD or on one's hard drive (or even its source code), or its execution? The two, importantly, are not the same even though they are both called Word: not only are they in different locations, one is a program while the other is a process." (Chun 2004: 48)

We are hence facing different 'states' of software: something that can be potentially transformed into object code, but is not yet compiled, something compiled, but not yet executed, and something that is actually being executed. In this context, the desire for a simple ontology of source codes or programmes or running software seems unsatisfiable. Such simple satisfaction will at least not occur if a philology of software wants to prevent itself from repeating the historic path (and blindness) of 19th century hermeneutic philology: in order to get hold of its object, texts, paradoxically it forgot about their materiality, replacing it with intangible concepts such as work. However, it might just be the process of compiling that could prevent us from repeating the philologic bug of romanticism's hermeneutics – seeing that it is only by turning a source code into something tangible to be executed by a machine that its purpose is fulfilled. Since such operativity (not to be mistaken with performativity) is a critical trait of the philologic object called software, it becomes questionable whether software's textuality is superposable with its source code.

As if German philologist Gunter Martens intended to formulate an extended notion of textuality, which helps to get hold of the two modes of software's existence, he wrote in 1971, pointing towards his concept of 'text dynamics': "It is not any single written form – the autograph, the print – that characterises the text, but rather an inherent tension directed towards an intentional formation" (Martens 1971: 169 [translated by the author]). While Martens describes the philologic significance of tension inherent to a text, this notion has a very material equivalent – regarding the case of a source code becoming running software: the states of voltage of a digital computer as condition of its data processing, and the very scene of digital materiality.

Martens' textual analysis of 'text dynamics' marked one significant turning point of German scholarship from a predominantly hermeneutic philology towards the materiality of the textual artefacts themselves. During most of the 20th century, models of textual criticism were based on either a specific author or work. This eventually inhibited establishing coherent methods for the concept of the infamous historical-critical editions. Aiming at a standardisation, Martens' considerations most importantly abandoned the then generally

3 "The word 'code' [...] oscillates between two different but closely, perhaps even undecidably related senses, *text* and *process*. This undecidable oscillation lies at the root of the problematic mode of existence of code." (Mackenzie 2003: 5 [original emphasis]).

accepted separation of the edited text accompanied by a critical apparatus containing mere variants. Instead, Martens argued for an equal representation of all textual versions, without singularising one version which, for example, may be considered by the editor to comply with the author's intentions. Consequently, all materialised versions of a work's genesis amount to its 'text'. Martens defines 'text', whose intentionality is to be represented qua philology, as "the sum of all single states that it runs through in the course of its genesis" (ibid.: 168f. [translated by the author]).

Assuming that the 'text' of software could also be described as the sum of all single 'states' that it runs through in the course of its genesis, it would then not just be the sum of all single states that its source code runs through in the course of its programming with and within the computer as a writing tool. Rather, the 'text' would also include all the single states that the code is transformed to and is running through during its intentional execution in the clocked steps of a machine process. This would be an implementation of what Martens called a text's "immanent movement" (ibid.: 169 [translated by the author]) – though not just in the form of a mere metaphorical movement, but in the materiality of a physical machine.

A Philology of Software

If we assume that the implementation in an operative machine becomes part of textuality, a philology of software would have to consider the following: Any scholarly edition of software is already preceded by something similar to an act of textual criticism. The compiler gathers a number of symbols from different textual occurrences and reconfigures them in order to make the outcome (machine) readable. However, it is crucial to consider the system of rules which the compiler is following while processing its material, that is how it 'reads' its data, and according to which rules it 'translates' them into another sign system. The question thus becomes one of different hermeneutics, of the interpretative nature of either human or machinic handling of textual objects. No critical edition of a literary text can ever be free of any hermeneutic intervention, because

"[...] a poetic text is an aesthetic phenomenon by nature and his representation therefore subject to the specific conditions of the aesthetic consciousness [of its editor]. [...] His involuntary subjectivity and the deviations of aesthetic beliefs cannot be eliminated from the outcome. Scholarly editing is interpreting." (Windfuhr 1957: 440 [translated by the author])

In contrast, the compiler can read, translate and reconfigure only according to syntactic and logic aspects. Completely devoid of erratic aesthetics, the compiler contradicts the very idea of subjectivity by its logic-syntactic nature. As a philologic endeavour, the act of compiling would thus implement a positivistic philology that was longed for in the 1970s by arguing for an uncoupling of philologic findings and their respective interpretation (cf. Zeller 1971: 51). Ultimately, the compiler eliminates from its operations any interpretation in a hermeneutic sense. Would it then be appropriate to say that software is an edited source code? Or that a scholarly edition of software would be achieved after compiling its source code?

These questions are related to a bigger debate about what qualities and capabilities can be advantageously ascribed to a machine. A common syntagma like 'the compiler reads/understands/translates' is uttered all too easily.[4] Reading, understanding and translating are performative (not operative) acts. Ascribing them only makes sense if there is the possibility of failure, that is: hermeneutics. Suggesting that 'the compiler reads/understands/translates' is thus a misguided anthropomorphisation of operative acts, simply because the compiler cannot misread, misunderstand or translate incorrectly. In contrast, thinking digital materiality would require a thinking of the act of machinic parsing: The parser, a programme responsible for the syntactic analysis of written input, is – unlike human beings – a finite-state machine that can therefore only process formal grammars. If the parser thus cannot interpret input, it does not make sense to say that a computer is 'reading'. Analogously, if – on the other side of the interface, where the user is located – to programme something is not to mean something, because formal languages are characterised by unambiguity, then there can be no misunderstandings, and speaking of 'understanding' does not make sense anymore. While it is necessary to clear up such anthropomorphic misunderstandings that will likely cloud the notion of digital materiality, it is debatable whether philology has to be limited to a strictly anthropocentric endeavour. Rather, it could be argued that traditional definitions of philologic endeavours like reading, understanding, translating – or even a philologic notion such as 'text' – are wrongly limited in their scope by ascribing them only to human beings (hence: humanities). Such argumentation was also brought up recently by Benjamin Bratton in relation to the question of (human) thinking and *Artificial Intelligence*:

> "We would do better to presume that in our universe, 'thinking' is much more diverse, even alien, than our own particular case. The real philosophical lessons of A. I. will have less to do with humans teaching machines how to think than with machines teaching humans a fuller and truer range of what thinking can be (and for that matter, what being human can be)." (Bratton 2015)

Likewise, a potential lesson of a philology of software could have to do "with machines teaching humans a fuller an truer range of what" (ibid.) philology can be.

The object and the methods of a 'philology of software' transcend or under mine philologic concepts of 'authorship' or 'text', of reading or translating, which originated in the hermeneutic spirit of German new humanism. What is entering the field as a philology of digital materiality could thus be called a posthuman-

4 Such an approach can even be found in the work of Kittler (1992: 147).

istic philology (cf. Hiller 2013: 156). Such agenda becomes even more urgent in the light of a global scientific paradigm called digital humanities, which might be in danger of not reflecting enough upon the materiality of its objects and its own methods – and thereby risks repeating the organisational blindness of 19th century hermeneutic philology. The critical challenge of a philology of software then lies within the question what, in contrast, it can and should accomplish: the configuration, preservation and transmission of the 'authentic' or the 'best' text (to invoke just two concepts of traditional philology)? Or rather to provide an executable – eventually even an executed – object of digital materiality?

Temporalities

Let us suppose once again that, paraphrasing Martens, it is also the single states which compiled software runs through in the course of its machine-execution that constitute its 'text'. Such operativity would then transcend the historic perspective of a classic genetic edition, which can only encompass the period of symbolic text production. Because, at least from a theoretical perspective, the validity of mathematical algorithms is timeless. Software's operativity thus evokes an ever present element, one that may not be conceived in the temporal dimension of historicity at all. Technical media do not obey the temporality of teleologic historical time. Rather they create very media specific temporalities in the course of their operative implementation (cf. Ernst 2013a, 2013b). In terms of a philology of software – whose temporality encompasses loops, jumps or recursions – the idea of a historical-critical edition of technical media is misleading, as such philology could not act upon the common assumption of a linear genesis which, for example, starts with an author's first sketches and comes to an end with a final authorised version.

As a physical quantity and in relation to the notion of history, the element of time that determines the complex ontology of technical media thus recursively becomes a critical parameter of editorial philology again. Textual criticism of software is time-critical. Philologist Herbert Kraft assumes that the task of textual scholarship lies within the documentation of texts as literary history, because "literary works are neither timeless nor time-dependent – they are historical" (Kraft 1990: 9 [translated by the author]). To be revealed within their edition is therefore "the genesis and history of texts as the production and transmission of that which becomes aware in the difference of utopia and reality: the shortcomings of real history in its historic occurrences, the constant contrast between the possible and the actual achievement" (ibid. [translated by the author]). How do technical media *s(e)ize* the space of this difference? What distinguishes a Turing machine from implemented software in a physical machine? To answer these questions would be the task of a philology of software and an aspect of digital materiality.

Kraft's definition of literary texts implicitly names the major difference between classic textual criticism and a philology of software. Symbolic media such as printed alphanumeric texts evoke history (instead of temporal processes), due to their hermeneutic potential which is put to effect differently in specific historical contexts: A literary text might be interpreted differently at different points in time. In contrast, signal-processing media, beyond any hermeneutics, defy teleologic forms of historicity and historiography. Software is mathematically timeless, since its source code will always and forever serve as a fixed prescription for a machine which can by processed by that machine in exactly one manner (i.e. if its code is correct): "[...] number series, blueprints, and diagrams never turn back into writing, only into machines" (Kittler 1999: xl) – and therefore, one could add, not into historically varying meaning. Kraft understands editions as different "forms of crystallization" of literary works and calls them in the words of Adorno a "site of the work's historical movement" (1990: 9 [translated by the author]). In contrast, a source code only has one timeless form of crystallisation without any historical movement. On the other hand though, in terms of materiality, a signal-processing medium like software is always time-dependent insofar as its execution, which ultimately fulfils its purpose, can only occur in the physical world and is thus dependent on a material implementation in a machine which necessarily is subject to objective time. Hence, in diametrical opposition to Kraft's understanding of literary texts as historical and thus neither timeless nor time-dependent, the latent tension within the 'text' of software could be identified as follows: it is not historical, but rather timeless and time-dependent at the same time. A historical-critical edition printed on paper could represent this temporality only by way of description, not (literal) prescription. "The critical approach to historicity is the potential of transmission", says Kraft (1990: 13 [translated by the author]). While that might hold true for any editorial philology of literary texts, the potential of philologic transmission of software is to be found somewhere else. Using such a critical approach to the complex temporality of digital materiality one may explore what constitutes digital media and what digital media constitute.

Materiality

To this effect, it might be insightful to take a closer look at the materiality of writing with a digital computer. Any critical edition of software would presuppose an understanding of the specifics of two different "writing scenes", as Rüdiger Campe (1991) called them[5]: on the one hand, the writing scene of a pen or a typewriter – a scene of performative materiality; and, on the other hand, the writing scene of a signal-processing machine – a scene of operative materiality. This difference may be better understood by thinking of a letter *A* that (1) was applied with ink on a piece of paper (coincidence/congruence of the sign and its materiality) or (2) a letter *A* that was written with a word processor

5 Campe's "writing scene" concept depicts the act of writing as "an unstable ensemble of language, instrumentality and gesture" (1991: 760 [translated by the author]).

of a computer. Philology traditionally conceives of the digital computer as a mere writing tool used by a – more or less – almighty author subject. The text that is being produced thereby is assumed to be a standalone object, materially separated from the writing tool. Two things are being obscured that way: A text produced by using digital computer technology initially does not exist as something that is materially detached from the writing tool – like a letter *A* that was applied on paper with a pen. Instead, it exists on the level of machine code as a stream of 0s and 1s stored in the computer hardware. Furthermore, even this bitstream of binary code is merely a symbolic representation of what is happening on the level of physical materiality, in the hardware of the machine: switchovers of sufficiently well distinguishable 'states' of electric voltage in the countless transistors of integrated circuits. Consequently, digital objects are ascribed three dimensions: physical, logical and conceptual.[6] What can be perceived on a screen, that is on the conceptual level, is thus, from a physical perspective, nothing else but a visualised section of the machine's current state of voltage: which is, ultimately, the tool of writing itself.

Thinking of the computer merely as a writing tool will eventually cloud the understanding of the writing scene that it constitutes and is an instrumental part of. What remains is a screen essentialism focused only on the interface of the writing scene and the machine output visualised thereon. In contrast, a philology of software has to consider the following: the writing scene's output, a text, and the writing tool that is actually producing the text are one and the same, as they take place in the same space of digital materiality. What does that mean for a philologic representation of software's textual genesis? By the end of the 1990s the textual geneticist Louis Hay, who coined the idea of a 'third dimension of literary texts' (Hay 1984 [translated by the author]), referring to the temporal process of their production, named one desideratum of digital editions: "In the long run, we should aim at a dynamic representation of writing, that allows for a chronological sequence to be shown on the screen and thereby visualise the 'third dimension' of time – even if it is just an electronic simulation" (Hay 1998: 77 [translated by the author]). Does this imply an electronic simulation of the 'text' of software as its genetic edition then?

If – as in the case of software – the scope of the textual genesis is questionable, the possibility of a genetic edition is at stake. Only a few years ago it was still argued that the computer as a writing tool puts an end to the era of genetic editing, because it does not leave any material traces of writing: "The first edition is all that remains of a first version of a work, and it is not possible to reconstruct its genesis." (Mathijsen 2009: 236) This fatal version of a philologic screen essentialism was vetoed by Thorsten Ries (2010) in reference to Matthew G. Kirschenbaum's digital forensics. Kirschenbaum (2008: 25-71) argues that every act of computer writing leads to a physical inscription that might not be conceivable with human eyes, but still leaves material traces that can be extracted.

6 See Kirschenbaum (2008: 3), referring to Thibodeau (2002).

Assuming that the phases of a text's genesis can be reconstructed perfectly by extracting these physical traces qua digital forensics, then Hay's wish for an electronic simulation of textual genesis becomes true: in a more encompassing way than the geneticist dared to ask for, since the employed writing tool is a discrete-state machine, and the text produced and stored on it is nothing more or less then one of its ever discrete 'states'. If all these phases can be extracted and reconstructed, it becomes possible to subsequently programme the genesis of a digital text as a simulation, reproducing one discrete machine state after the other, that is, in the case of the computer writing scene, keyboard hit after keyboard hit. To object that such a simulation would not represent the factual moment of textual creation via an author would only demonstrate a philologic screen essentialism, originating in the familiar logic and materiality of continuous handwriting. Under conditions of digital materiality though, this moment, if at all, does only take place on this side of the interface – not where the act of material inscription is taking place.[7]

The consequences for a philology of software and its relation to the object it is supposed to represent could not be more severe though. Again: the critical edition in the form of an electronically simulated 'text' genesis would also have to incorporate the executed programme as an integral part of that genesis. But a simulation of the 'text' of software that also incorporates the discrete states of execution – which eventually would be nothing else then the actual execution – would not be a philologic representation of the software's 'text'. Rather it would be the running programme itself. Eventually, the medial and material gap between a philologic object and its representation, which is the condition of possibility of all editorial philology, would be repealed. The possibility of editing (Turing-)completely turns out to be editing digital materiality's essential contradiction. Not being able to alternatively print its object on paper, the challenge for a philology of software will be how to deal with this aporia.

Conclusion

"It is one of the metaphysical issues of modern philology, what editorial philology is actually concerned with: the text as a virtual entity or rather its mere material realization, a printed book, a manuscript or any other kind of medium." (Joost 2000: 359). What the textual critic Ulrich Joost expressed as the philologic question is unintentionally also relevant to thinking about a philology of software. Software, one could say, is never at one's disposal other than in its material implementation which enables its operativity. A final answer to the metaphysical question of philology could thus be its liberation of all metaphysics, leaving behind nothing but pure physics of digital materiality. A philology of software

7 "As one knows without saying, we do not write anymore. [...] Nowadays, [...] man-made writing passes instead through microscopically written inscriptions, which, in contrast to all historical writing tools, are able to read and write by themselves." (Kittler 1997: 147)

therefore first of all must not reduce the differences of symbolic and signal-based data processing to their mere surface effects. Consequently, where the writing tool is said to be reading itself, a philology of software cannot just rely on the cultural technique of reading in its traditional sense, may it be close or distant, in order to investigate its objects.

Revisiting the source code presented in the beginning of this text might enable the realisation that it somehow prescribes the script following it. What cannot be comprehended this way is however what the source code, if at all, does with the text as an executed programme. Neither the text nor the source code are thus the 'text' of this piece of software. Printing this source code as part of this text – a piece of code that could also be compiled as object code possibly making a machine print the text – ultimately illustrates nothing more than the fact that this object of digital materiality can never be conceived in both its dimensions at the same time, only exhibiting its epistemic uselessness regarding the question of a philology of software.

At the beginning of the 1970s, when German editorial philology started to turn towards the materiality of its objects, Martens and Zeller emphasised the impossibility of writing a comprehensive manual of textual criticism: "Definite conceptions about the tasks and the ends of critical editions that could be based solely on a technological instruction, do not exist yet in contemporary philology." (Martens/Zeller 1971: VII) It is arguable whether such instructions for the scholarly editing of literary texts are available now – or if they ever will be. In terms of what could be called the most contemporary philology though, a philology of software, whose initial questions were to be illustrated in this text, it might be possible that the technological instruction of philology that was once yearned for is now prescribed – and that is not to say: programmed – by its object itself: signs o' the times of digital materiality.

References

Berry, David M. (2011): The Philosophy of Software. Code and Mediation in the Digital Age, Basingstoke: Palgrave Macmillan.

Bratton, Benjamin H. (2015): "Outing A. I.: Beyond the Turing Test", February 23, 2015 (http://opinionator.blogs.nytimes.com//2015/02/23/outing-a-i-beyond-the-turing-test).

Campe, Rüdiger (1991): "Die Schreibszene, Schreiben." In: Hans Ulrich Gumbrecht/K. Ludwig Pfeiffer (eds.), Paradoxien, Dissonanzen, Zusammenbrüche. Situationen offener Epistemologie, Frankfurt am Main: Suhrkamp, pp. 759-772.

Cayley, John (2002): "The Code is not the Text (unless it is the Text)", September 10, 2002 (http://electronicbookreview.com/thread/electropoetics/literal).

Chun, Wendy Hui Kyong (2004): "On Software, or the Persistence of Visual Knowledge." In: Grey Room 18, pp. 26-51.

Ernst, Wolfgang (2013a): Gleichursprünglichkeit. Zeitwesen und Zeitgegebenheit technischer Medien, Berlin: Kadmos.

Ernst, Wolfgang (2013b): Chronopoetik. Zeitweisen und Zeitgaben technischer Medien, Berlin: Kadmos.

Fuller, Matthew (ed.) (2008): Software Studies. A Lexicon, Massachusetts: Leonardo Books.

Hagen, Wolfgang (2006): "The Style of Sources: Remarks on the Theory and History of Programming Languages." In: Wendy Hui Kyong Chun/Thomas Keenan (eds.), New Media, Old Media: A History and Theory Reader, New York: Routledge, pp. 157-175.

Hay, Louis (1984): "Die dritte Dimension der Literatur. Notizen zu einer 'critique génétique'." In: Poetica 16, pp. 307-323.

Hay, Louis (1998): "Drei Randglossen zur Problematik textgenetischer Editionen." In: Gunter Martens/Hans Zeller (eds.), Textgenetische Edition, Tübingen: Niemeyer, pp. 65-79.

Hiller, Moritz (2013): "Diskurs/Signal (I). Literaturarchive nach Friedrich Kittler." In: Friedrich Balke/Bernhard Siegert/Joseph Vogl (eds.), Mediengeschichte nach Friedrich Kittler, Paderborn: Fink, pp. 147-156.

Joost, Ulrich (2000): "'Als müßte ich es mir übersetzen' – Prolegomena zu einer editionskritischen Untersuchung der deutschen Zweischriftigkeit." In: Rüdiger Nutt-Kofoth/Bodo Plachta/H. T. M van Vliet/Hermann Zwerschina (eds.), Text und Edition. Positionen und Perspektiven, Berlin: Erich Schmidt, pp. 353-368.

Kirschenbaum, Matthew G. (2008): Mechanisms: New Media and the Forensic Imagination, Massachusetts: MIT Press.

Kittler, Friedrich (1997 [1992]): "There Is No Software." In: Friedrich A. Kittler, Literature, Media, Information Systems: Essays, ed. and trans. by John Johnston, Abingdon: Routledge, pp. 147-155.

Kittler, Friedrich (1999): Gramophone, Film, Typewriter, Stanford: University Press.

Kraft, Herbert (1990): Editionsphilologie, Darmstadt: WBG.

Mackenzie, Adrian (2003): "The Problem of Computer Code: Leviathan or Common Power?", March 2003 (http://www.lancs.ac.uk/staff/mackenza/papers/code-leviathan.pdf).

Marino, Mark C. (2006): "Critical Code Studies", December 4, 2006 (http://electronicbookreview.com/thread/electropoetics/codology).

Martens, Gunter (1971): "Textdynamik und Edition. Überlegungen zur Bedeutung und Darstellung variierender Textstufen." In: Gunter Martens/Hans Zeller (eds.), Texte und Varianten. Probleme ihrer Edition und Interpretation, München: Beck, pp. 165-201.

Martens, Gunter/Zeller, Hans (1971): "Vorwort." In: Gunter Martens/Hans Zeller (eds.), Texte und Varianten. Probleme ihrer Edition und Interpretation, München: Beck, pp. VII–X.

Mathijsen, Marita (2009): "Genetic Textual Editing: The End of an Era." In: Gertraud Mitterauer/Ulrich Müller/Margarete Springeth/Verena Vitzthum (eds.), Was ist Textkritik? Zur Geschichte und Relevanz eines Zentralbegriffs der Editionswissenschaft, Tübingen: Niemeyer, pp. 233-240.

Ries, Thorsten (2010): "'Die geräte klüger als ihre besitzer'. Philologische Durchblicke hinter die Schreibszene des Graphical User Interface. Überlegungen zur digitalen Quellenphilologie, mit einer textgenetischen Studie zu Michael Speiers ausfahrt st. nazaire." In: Editio 24, pp. 149-199.

Thibodeau, Kenneth (2002): "Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years." In: Council on Library and Information Resources (ed.), The State of Digital Preservation: An International Perspective (http://www.clir.org/pubs/reports/pub107/thibodeau.html).

Windfuhr, Manfred (1957): "Die neugermanistische Edition. Zu den Grundsätzen kritischer Gesamtausgaben." In: Deutsche Vierteljahresschrift für Literaturwissenschaft und Geistesgeschichte 31, pp. 425-442.

Zeller, Hans (1971): "Befund und Deutung. Interpretation und Dokumentation als Ziel und Methode der Edition." In: Gunter Martens/Hans Zeller (eds.), Texte und Varianten. Probleme ihrer Edition und Interpretation, München: Beck, pp. 45-89.