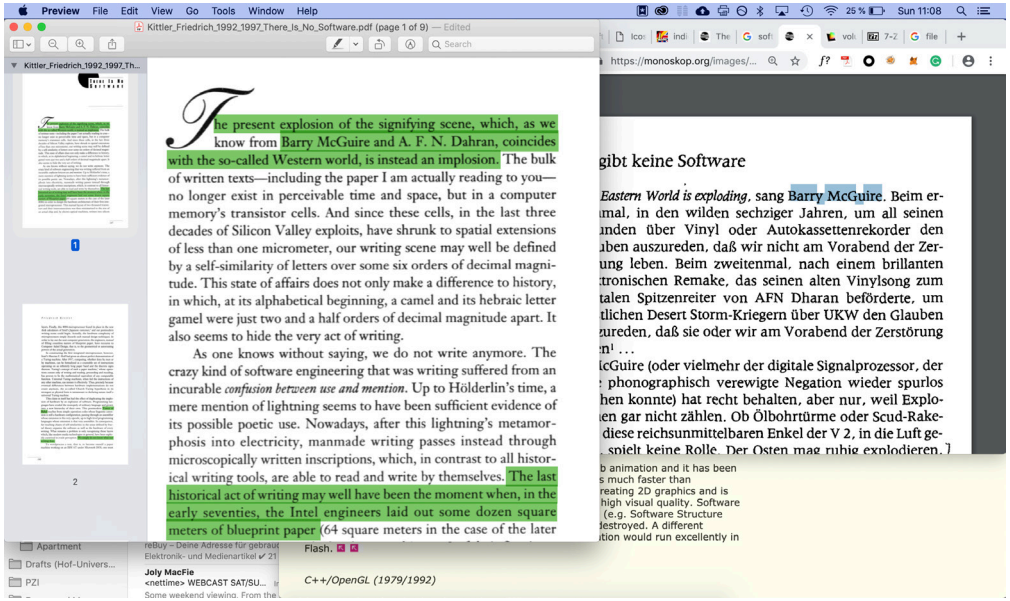
Software is not visible,
Software is performing.

Software provokes.

Software tells.
Software fails.

Software is imperfect.

Software is poetic.



Software has taken command in our daily lifes. It is omnipresent and most of our recent world would come to a halt without it. Software has become so ordinary, that it is often overlooked. Software is taken for granted while it is increasingly entangled in our life and constantly takes over new tasks. Our computers are becoming smarter through new kinds of algorithms. This leads to new challenges in understanding software – not only from a scientific point of view but also from a cultural, political and social perspective.
And so software has also found its way into the art and vice versa, but there are still gaps in the relation between both of them. I think that the interaction between software and art can help both of the disciplines to improve.

The question that I am asking is: what can an artistic method for researching the processes and relations of software look like? How can we look critically into the software that we use on a daily basis. The current perception and use of software are significant parts of this research. Especially in contrast to the original culture around software, that included hacking and that required every artist to write their own software.

This essay explores the multiple layers of software with a special focus on the dependencies that arise around and through software. Software is made up of several layers.
The code, the execution (the executable), the output, the user interaction. Code is a well researched topic, also in the arts, as well as the output has a long history in the arts. But the part of execution has yet mainly been seen as the functional part. In the following essay I want to show the potential artistic use in the processes of software.

During the research of this project, I found myself returning to the same essay over and over again, drawing inspiration from and following up on the various issues touched upon by Kittler. I uncovered a great variety of controversies surrounding the creation, execution and use of software.
Furthermore, I realized that the more research I did on software and its implications for our lives the more aware I became of the soft-

ware that I have been using. I started observing my own attitude towards various applications that have been shaping my life and work everyday and started questioning many functions and backgrounds of software that I had viewed as a given before.
I became an Ethnographer of my own work in progress. I realized that my own behaviour and everyday occurrences in the interaction with software reflected what I was reading in research papers and articles on my screen and vice versa. Kittler therefore serves as a point of departure for different controversies around software. This will lead me to the arts, and why I think art might provide possible approaches towards these different topics.

The first part of my work will be an ethnographically inspired examination of the interaction with my computer whilst reading Kittler's essay »There is no software«. The text will unfold on two different levels: On the one hand I am describing the process of reading while interacting with the software I need to do so. On the other hand there will be interventions to critically reflect on various concepts touched upon. These interventions refer to either Kittler's text itself, or to the software that I am using.

In the second part I will describe how art provides different frameworks to approach the different aspects I pointed out in the first part.

Why this method?
The detailed description of reading digitally makes the different software that is being used visible. Through that the software can be observed while at work. Next to this it is a great chance to revisit the text of Kittler. This method also allows for new encounters and associations with software, that will help to recognise the different agents at stake when thinking about the processes of software and the involvement of art with it.

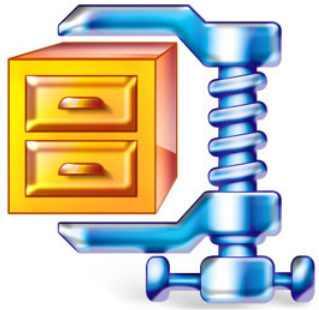
Why »There is no software« by Kittler?
This text very early became one of the key texts of my interest and research. The text offers a great source for thinking about software today. Because in his essay from 1992 he is actually not negating the existence of software, instead he wants to emphasise the materiality, that is being neglected in his opinion.
This is a big tension that we can also recognize in computation today. Even if we do not neglect software, it becomes more and more invisible, we imagine software mostly through metaphors. Workflows are so seamless it seems almost like there is no software.

I am reading “There is no software” by Friedrich Kittler. I downloaded the pdf file to my computer using the browser. The file is now stored in my file-system, which I can view in a representational view by opening the file explorer. I double click my way through the folders until I end up in the Downloads folder, where the newly downloaded file is placed in a list view amongst others. The file is called “Kittler_Friedrich_1992_1997_There_Is_No_Software.pdf”. I hover the small bar with the title, the operating system proposes to open the file with the document viewer and so I do, by gently double clicking the left mouse button. Within seconds a new window appears putting the file manager window into the background and foregrounding the title page of the pdf framed by small icons and scrollbars. I click to enlarge to fullscreen and start to scroll down till the first lines of text appear. I zoom out pressing CTRL and - twice. Next, I start reading the first sentence. “The present explosion of the signifying scene, which, as we know from Barry McGuire and A F. N. Dahran, coincides with the so-called Western world, is instead an implosion.” Barry McGuire? I hover the name, press the mouse down and drag from B to e. The text tints white with a blue background. The release of the mouse button is followed by pressing CTRL C. I switch to the browser, which still shows the download page of the PDF. I paste the name into the search bar and press enter. The search engine shows a list of results – one video, this must be it. As the link reacts to my hovering, I click on it and with a short flickering I end up on youtube. Without any action required the video starts and the speakers sound: “The eastern world it is exploding”, to which Kittler must have referred.

Compression

The implosion and explosion can well be seen on different levels of software. While the complexity and interplay of different technologies are exploding, the visibility and the potential for understanding are imploding. Increasingly better software brings great advances in e.g. computer vision, but at the same time it becomes harder to understand. The potential of having more sophisticated technology may come at the risk of blurring the understanding.

At the same time these highly complex algorithms require more hardware and even better processors.
The implosion of files a very well used method in the form of compression. Compression needs software that is able to rearrange the bytes of files using various algorithms, for the sake of file size. Smaller files can be stored easier and have advantages for transmitting. But his can have different implications.



IMG 2: REFERENCE

It is a method to circumvent the physical limitations (to some extend). This means that files can be stored with very little storage available.
Other than that, we produce increasingly bigger files, because cameras output high-resolution images, we can gather more data, scan better and display highly sophisticated websites. But how does this effect us in the real world? Unlike the imagination that the digital is de-materialising, the processing of big files for instance is consuming much energy (https://solar.lowtechmagazine.com/2018/09/how-to-build-a-lowtech-website.html).
Therefore some websites like the lowtechmagazine are developing different methods on how to host low-energy web pages. They are using solar panels and produce their websites in a way that makes the site very light in terms of files that have to be transmitted. So from this case we can see, that compression can have more effects than assumed. It is these small nuances that make software a powerful tool to think about current cultural topics.
This lightweight approach gives reason to think about different aspects of how websites are being served and how they are built.

I stop the video by clicking onto the face of the singer and a

smoothly appearing pause sign inside a circle signals the success of my action. I change back to the document viewer by clicking on the window that got hidden in the background by the browser.
The words I read are displayed with a grained border presumably caused by the scanning process. As I read on, my t-shaped cursor, follows the lines of the text.
I continue with the next sentence. »The last historical act of writing may well have been the moment when, in the early seventies, the Intel engineers laid out some dozen square meters of blueprint paper« (Kittler, 1992).

Dependencies

> With its increasing speed, computation fosters itself while depending on the previous version of its own. The same holds true for software. Therefore we can recognise a spiral of dependencies and influences that includes humans and machines. After the first hardware was able to draw new, even smaller hardware than it would ever have been possible with paper and pen, the system of hardware design became dependent on itself. (Kittler, 1992)
> This means the next generation of hardware is always enabled by and relying on the previous version, making it possible to create even smaller and more complicated parts. The same can be found in the culture of software development. Software can only be built with software: software that enables to write the program code, software that compiles the code into machine readable binary-code and an operating system that executes it. This also means that nearly every program relies on other ones, requiring users to pre-install specific versions of software in order to run the program. If one single component of this dependencies breaks, many other programs will be effected.
> The dependence on companies that produce software is great. If the company decides to discontinue their software, the user is directly influenced by it as he can not use his software anymore. For instance, when Microsoft shut down one of their scripting languages, many companies that relied on it suffered. (Close to the machine, 105)
> Software is growing with time. As Ellen Ullman mentions, many programmers will work on a system until it becomes nearly impossible to understand. Still, you will have to keep it running making it an obsolete system, in which you can hardly change anything (Close to the machine, p.117). These kind of dependencies tell their own stories and are rarely clearly visible.
>
> »We shape our tools and, thereafter, our tools shape us« (https://medium.com/@freddavis/we-shape-our-tools-and-thereafter-our-tools-shape-us-1a564cb87484) says a famous quote by John Culkin from 1967. But if we look at the dependencies of software one could also say: we shape tools and these tools shape new tools again. Transferring this idea to the notion of software as a cultural object, the interrelation between shaping and being shaped could be formulated as follows: software creates and influences culture, and therefore this culture shapes new social conditions under which the construction and use of software itself is altered. This might become clear when looking at the example of software-hacking. The distribution of proprietary software with Digital Rights Management (DRM) lead to multiple groups cracking and circumventing software limitations. These cracks are then distributed as new software.
> The original culture of software was actually build around open source culture. Early software production was very dependent on this openness. Without open source it would not have been possible to develop software further. (Aymeric, p.9)

I further follow the dark pixels on the screen to the roaring sound of the computer. It is not clear whether the ventilation sound is triggered by the hardware or the software, which is causing the CPU to overheat. Kittler is writing about how the language gets ab-

stracted from high-level, human readable words, to assembler code, that is being translated into non readable machine code. As Kittler talks about this »postmodern Tower of Babel« (1992, p.148) I realise how my windows have started to built up like a tower. The document viewer on top of the browser on top of the settings on top of the mail program and so on.

Framework culture

Programming languages are based on other programming languages in order to make the code easier to write and read. Low-level languages are very close to the actual machine processes and therefore very complex to write. This is why high-level languages were constructed to translate this elaborate processes into human readable concepts and language.
In addition to that programmers often rely on third party frameworks, which provide functions that are very convenient to implement. Instead of having to write the code themselfs, they just have to put one line of import. Therefore the whole set of tools provided by the so called library becomes available for the use of the programmer. The process of using frameworks often obscures the actual algorithms. For example it can be quite challenging to create a machine learning algorithm from scratch but frameworks like »keras« or »tensorflow« make it accessible. The problem is that the programming syntax is very close to human language, which makes it hard to comprehend the actual code. Thus it is harder to change functions that are underneath the layer of the framework-interface. (Cox, 2007, p.153)
Furthermore the different programming languages favour different concepts of language and writing as well. (Cox, 2007, p.153) So the choice of programming language already determines a certain style of writing. And, because language significantly shapes our imagination, the choice of programming language also influences our understanding of software. Although scripting languages are very popular right now, it should not and can not replace low-level programming.

»High-level programming approaches can be very successful in achieving certain ends, but the very imposition of higher-level constructs and metaphors also limits awareness of how code operates in and for itself and what may be achieved through that. Arguably it is the changes in low-level systems that have provoked the biggest paradigm shifts, such as the development of binary computation and Turing machines [...]« (Yuill 2004)

To me this also means that an active engagement with different levels of programming is necessary to reflect important aspects of computation. A critical practice around software should therefore not only focus on one specific programming language. This helps to free yourself from the dependencies stated above and enables you to engage on different layers, not only the surface.

[→ Argument: more brutalist software → make software more durable / more accessible]

I continue in the text, and while Kittler is buying a commercial version of WordPerfect, I remember my old copy of Word that still must by on my hard drive somewhere. I go through the folders of my applications folder of my second partition scan through all the apps, that I probably haven't used for month. I follow the alphabetical order of the list view and after N, appears a folder called Microsoft. I double click on the icon of an orange folder and end up in a grid view, containing 6 files and some folders. Inbetween them: word.exe. I can't open it on Linux.

I am a consumer not a user

Nowadays the software that is required to use a machine comes pre-installed and ready to use. Software can be downloaded from centralised marketplaces: App Stores. This causes an immense dependence on the producers. These producers have developed an infinite selection

of apps for everything. This is another example for the »explosion« of software that was previously mentioned. This flood of applications causes software to become a mundane occurrence. The danger of that is that we take software for granted. When we have a problem, there is an app for it. Nobody thinks about the possibility of editing software and adjusting it to one's need. This is not only because most of the time it is not possible to edit the software due to DRM but also because the average user is not a user anymore. Rather people are being educated by companies to be consumers instead of users let alone creators. It is in the companies interest to make their clients dependent on their product. Therefore companies are not interested in opening up their products, but they are instead locking it up. They are then slowly feeding their clients with updates and new fancy features. This is great for users who just need to get their job done and who want to be in contact with technical struggles as little as possible. On the other hand it means that firstly, the imagination of software is dictated by companies and secondly that IF you want to engage with your software you can't do so. You can't look at the source code, reuse parts of it and you can't modify the program to your needs.

Of course there is also another end of the spectrum: hackers and creators with custom software and completely denying any use of commercial software. This movement also provides a great source for discussion about software.
The problem is that the average user is not happy about struggling to install what they need before they can actually write something. There are also other kinds of software, that embrace the user as an active agent, while still enabling an easy use on the surface. For example the mediawiki software allows for easy editing on the browser, while still providing an infrastructure to easily extend the functions. (source)

»The accompanying paperware« – wait, which paperware? Where is the manual of my document viewer? I move my mouse towards the options on top of the window and click on help. A small window opens, displaying a table of contents. »How to use it« »Find text in documents«… A page containing hyperlinks for different sections. It is probably the first time I ever entered this space of the program.

The manual of most programs is part of the software. Actually, the manual is software. The handbook does not come in a physical form anymore. Just as the software does not ship on Floppy or CD-ROMs. Software is a download, so it never really enters the physical space anymore and thus, it becomes even more abstract. Through the handbook, the software manifests itself as a tool. A tool, that has certain functions and the manual describes how to use those functions correctly. Nowadays, the handbook often constitutes a space that stays undiscovered. If we want to consider software as an artistic material, the handbook can also gain new functions as a description, as a space for thoughts. The handbook was also used as a metaphor in the readme festival 2006, to guide visitors through an exhibition of software. Software often remains invisible in its functions and statements, so it is necessary to describe what each exhibit is doing.
Software can be so abstract, that the way how software affects people is often through the metaphors it uses. What we remember is the animal on the start-screen, not the algorithm that it uses. For an artistic engagement I think it is important, to carefully examine the different parts of software and then reflect on their use – like the metaphor of the user manual.

I close the help, and find my way back to the text. In the meantime, Kittler turns towards his punchline: There is no software.

Even though software is depended on hardware, it does not mean that there is no software. A deeper engagment with software also means taking software seriously. Even

though it might be argued that software is only the representation of machine operations, it is important to acknowledge software as an independent object of study. Even though Kittler was arguing that there is no software and it is intrinsically connected to its hardware, Cramer points out that »if any algorithm can be executed mentally, as it was common before computers were invented, then of course software can exist and run without hardware« (2002). Following this argument it points to the idea of software in a very conceptual way, not only defining software as a program that is running on a certain hardware. All layers of diminishing abstraction on top of hardware deserve attention. Still it is important to recognise both of the perspectives for their importance – the materialistic and the cultural / political. Anyway, there is no clear border between software and hardware. Where does software begin and Hardware end? Is it when the Code is being compiled or is it when the machine code is transformed into electrical signals?

There is certainly a tension between the development of software and hardware. The hardware limits the software. We can not build applications that run faster than the hardware. Machine Learning algorithms for example need a lot of resources to calculate their models. This means that effective research with this technology is only possible with sufficient hardware. Even though software can be seen as a conceptual good, it is impossible to execute it only mentally, especially when using very complicated algorithms. Software is only effective through its execution, its performance.

»First, on an intentionally superficial level, perfect graphic user interfaces, since they dispense with writing itself, hide a whole machine from its users.«

The user interface enables a convenient way to display software (or at least parts of it). This representation is however only an interpretation of what the designer thought is the best way to display it (Hadler, 2016, p.7). At the same time it looks like this user interface is the only truth that the program holds. It does certainly not become obvious that this interface is not neutral. The GUI instead hides. It hides the processes, a lot of functions, the source code, the possibilities, the decision it takes for you.

The need for a human approach to software also becomes visible from the great use of Graphical User Interfaces. The so called GUI, is not part of the original imaginary of computation, where commands were being filled in via a command line. But today's average user is only surrounded by software displayed via a "window", encountering the terminal only by chance. Not only does the GUI simplify commands into buttons and mouse-actions, but also does it make software more human. A button that has a 3D effect (Software Studies → Button), the on/off function is displayed via a switch, the mouse transforms into a hand or the form that looks like a letter, which off course you fill in by pressing a pen symbol. This is also known as skeuomorphism. It means that objects of the real world are being used for representing digital functions or interface objects. Humans anthropomorphize and use metaphors to communicate the complexities of a less well known domain (the digital) via the vocabulary and concepts associated to a well known domain (the physical world). The skeuomorphism in GUIs is a good example for that.

As I go further in Kittlers text, focusing on the text as my mail software wants to interrupt me with some notifications about incoming mails. I click them away. Kittler is writing about how computers are writing and reading themselves. I want to copy this part into my notes. I drag the mouse from »in contrast« to »read and write by themselves« and as the text tints, the layer of text reads: »in contnast to all histor- ical writigtools, are able to read and write by thenvselves« (1992, p. 147). My machine has read the text before

me – not only once. Actually the text has probably been written and read many times before I opened it. The computer had read the document for words using Optical Character Recognition and even made its own interpretation. That explains why the selected text is wrong, because the program misinterpreted some of the characters. Together with this not incorrect version of the text, it got written again to the memory. Then another time the text was read once again – into the working memory, when I opened it with the document viewer.

Glitches and non-functional software

Software always comes with a dedicated function or purpose. Although software is meant to be used, to be executed, there are also other important layers that are not only functional (Goriunova and Shulgin, 2004, p. 161). Yet, we can get a spark of what execution of code means and how software really acts and performs when it fails or when it is taken out of its context (Understanding Computers Source). So in the following I want to argue that for a serious engagement with software it is also necessary to look at the non-functional and the stuff that is in-between the pixels and conducting paths. Software is primary made to function, but what if software fails or malfunctions on purpose? What, if software has no function?

While The Alliance for Code Excellence imagines »[a] world where software runs cleanly and correctly as it simplifies, enhances and enriches our day everyday life is achievable« (Constant, 2018, p.11) I argue that the malfunctioning of code can also be something positive that is revealing and holds a value.
The wrong character recognition as visible from the text above, can show how the algorithm works. The mistaken “m” for “rn” shows that the algorithm might work with visually comparison and has probably not recognised the gap between “r” and “n” – due to the grain of the text. This consequently gives a clue, that the algorithm doesn’t have an idea about the context of words. Otherwise it would have figured out that some words are not correct English words.
Furthermore I am thinking about an unstable setup, where the user knows that there is a potential for crashes. It means that engagement is undeniable. At the point when it crashes you will be able to get a glimpse of the inner workings of software. In contrary if you rely on the system and you have no knowledge of how it works, you will be unable to fix it in case of failure. This means that software shapes our behaviour and software itself can be engaging or not.

Other then malfunctioning software is also taken out of its context when it is used wrongly. The unintended use of software can arise from an uninformed user or an user trying to stretch the potential of the functions. Some people are collecting misuses of software and operating systems online and it can be quite entertaining (Thought Catalog, 2016). The open source license embraces this fact of re-using. By giving open access to the source, it also gives the freedom to reuse code for other purposes.

The Imperfection of software
Digital System are often considered to be pixel-perfect. But instead also digital applications become unstable as they fail. Software can even have the same noise as non-digital objects have. When Casey Reas wrote about the new Processing he pointed out the precision of computers compared to similar art-forms like Sol LeWitt practiced it. »[…] [M]achines can draw lines with absolute precision so all the imperfections in a physical drawing are removed, giving the rendering different characteristics than those intended by LeWitt.« (https://artport.whitney.org/commissions/softwarestructures/text.html) In reality it turned out that after a few month processing produced the same inaccuracies

(glitches) as a drawing by LeWitt would show. This was due to updates and changes in the language.

I change from the document view into the writing program Libre Office, where I store most of my notes. With a single click on the icon, no keystroke required, the execution starts and the start screen appears. Many process get triggered by this simple action and the computer follows its instructions, which I do not know – and not even see. But not with ease this time. The only thing that I can occupy right now, that the process must have »stuck«. As my mouse indicates with a spinning motion, I am unable to continue. I am unable to change the program, I am stuck, just like my program. I try clicking on the icon, again and again, as if my actions would trigger the program to finally make it. It is as if I want to tell the program to try harder by clicking harder. Once again I try to encourage the app, by clicking somewhere randomly on the screen. I give up. I have had this before, so I know how to act. »sudo kill«. I change to the terminal, type sudo kill libreoffice. I give my permission and happily I can see the terminal taking action. With a flicker the startup screen that was stuck disappears, freeing me and my cursor from redundant spinning. I try restarting the program and hope, that the crash was only due to difficult circumstances, maybe just something »got stuck«.

> The perception of software is anything but neutral. Software tells stories, through (1) its metaphors, (2) through its contents, (3) through its performance.
>
> The digital medium offers new ways of telling stories. This becomes obvious not only due to different structures, like the form of the database as Lev Manovich points out, but also because of the different modes of intervention software takes in our life. (Manovich, 1999) Furthermore the medium keeps evolving at inexorable speed and so does software, leaving space for new ways of how to tell and what to tell about computation.
>
> That humans tend to anthropomorphize not only their surroundings but also computers and technology in general has been a well researched topic among computer sciences & psychology. In addition to that humans have a vivid and diverse imagination about processes that are invisible. This includes software. Often digital media black-boxes certain processes and therefore provides a lot of space for imagination and narratives that can be constructed around it. (Finn, 2017, p.229) These stories in applications and around make technology more understandable, but can also be source for misconceptions. A current example seems to be the fear of singularity after machine learning enables applications to »magically« generate or label images. The gap between the real potential and the imagination about it is big. I don't want to support an uncritical or blind approach towards technology – I think it is important to be realistic, critical and playful equally with these algorithms, only then turns engagement into insight.

Among others, "The media equation" had shown, that we as humans consciously and unconsciously anthropomorphize computers (Reeves and Nass, 2003). Narratives have been used for the purpose of marketing and there have been attempts to create relatable stories within applications. A well known example is Joseph Weizenbaum's Eliza, a digital application, that acted as a therapist, chatting with the user. This piece of software gave impressive proof of how humans anthropomorphize even simple digital applications. (Expressive Processing, p.27). Tech giants have put great effort in implementing relatable characters into their systems, e.g. voice assistants. An assistant, that is helpful and funny, that gathers you data with great pleasure. But in the past there have also been unsuccessful attempts to add anthropomorphizing elements to programs, only to remind quickly about Microsoft's famous Clippy (https://www.artsy.net/article/artsy-editorial-life-death-microsoft-clippy-paper-clip-loved-hate).

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?mso-application progid="Word.Document"?><w:wordDocument xmlns:w="http://schemas.microsoft.com/office/word/2003/wordml" xmlns:wx="http://schemas.microsoft.com/office/word/2003/auxHint" xmlns:o="urn:schemas-microsoft-com:office:office" xmlns:aml="http://schemas.microsoft.com/aml/2001/core" xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882" xmlns:v="urn:schemas-microsoft-com:vml" xmlns:w10="urn:schemas-microsoft-com:office:word" xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0" xml:space="preserve" w:embeddedObjPresent="no">
<o:DocumentProperties xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><o:Title/><o:Subject/><o:Keywords/><o:Description/><o:Category/><o:Author/><o:LastAuthor/><o:Manager/><o:Company/><o:HyperlinkBase/><o:Revision>1</o:Revision><o:TotalTime>0</o:TotalTime><o:LastPrinted/><o:Created>2019-02-11T18:16:28.393800095Z</o:Created><o:LastSaved>2019-02-11T18:16:55.865428995Z</o:LastSaved><o:Pages>1</o:Pages><o:Words>25</o:Words><o:Characters>174</o:Characters><o:Paragraphs>1</o:Paragraphs></o:DocumentProperties><o:CustomDocumentProperties xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><o:Editor dt:dt="string">LibreOffice/5.4.4.2$MacOSX_X86_64 LibreOffice_project/2524958677847fb3bb44820e40380acbe820f960</o:Editor><o:Language dt:dt="string"/></o:CustomDocumentProperties>
<w:fonts><w:defaultFonts w:ascii="" w:h-ansi="" w:fareast="" w:cs=""/><w:font w:name="Executive-55Reg"><w:family w:val="Roman"/><w:pitch w:val="variable"/></w:font><w:font w:name="Liberation Serif"><w:family w:val="Roman"/><w:pitch w:val="variable"/></w:font><w:font w:name="MinionPro-Regular"><w:family w:val="Roman"/><w:pitch w:val="variable"/></w:font><w:font w:name="Liberation Sans"><w:family w:val="Swiss"/><w:pitch w:val="variable"/></w:font><w:font w:name="Arial Unicode MS"><w:family w:val="System"/><w:pitch w:val="variable"/></w:font></w:fonts>

<w:lists xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:listDef w:listDefId="0"><w:lvl w:ilvl="0"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%1"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="1"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%2"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="2"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%3"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="3"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%4"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="4"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%5"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="5"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%6"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="6"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%7"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="7"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%8"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl><w:lvl w:ilvl="8"><w:start w:val="1"/><w:nfc w:val="255"/><w:lvlText w:val="%9"/><w:lvlJc w:val="left"/><w:suff w:val="Nothing"/></w:lvl></w:listDef><w:list w:ilfo="1"><w:ilst w:val="0"/></w:list></w:lists>

<w:styles>
<w:style w:styleId="default-paragraph-style" w:type="paragraph" w:default="on"><w:name w:val="default-paragraph-style"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing/><w:ind/><w:widowControl w:val="on"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Liberation Serif" w:h-ansi="Liberation Serif" w:fareast="Arial Unicode MS" w:cs="Arial Unicode MS"/><w:sz w:val="24"/><w:lang w:val="en-GB"/></w:rPr></w:style><w:style w:styleId="default-table-style" w:type="table" w:default="on"><w:name w:val="default-table-style"/><w:tblPr><w:tblInd xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:w="0" w:type="auto"/></w:tblPr></w:style><w:style w:styleId="Standard" w:type="paragraph"><w:basedOn w:val="default-paragraph-style"/><w:name w:val="Standard"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing/><w:ind/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"/></w:style><w:style w:styleId="Heading" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Heading"/><w:next w:val="Text_20_body"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:before="239.841" w:after="120.204"/><w:ind/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Liberation Sans" w:h-ansi="Liberation Sans" w:fareast="Arial Unicode MS" w:cs="Arial Unicode MS"/><w:sz w:val="28"/></w:rPr></w:style><w:style w:styleId="Text_20_body" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Text_20_body"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:line-rule="auto" w:line="288" w:before="0" w:after="140.049"/><w:ind/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr></w:style><w:style w:styleId="List" w:type="paragraph"><w:basedOn w:val="Text_20_body"/><w:name w:val="List"/><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"/></w:style><w:style w:styleId="Caption" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Caption"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:before="120.204" w:after="120.204"/><w:ind/><w:widowControl w:val="off"/><w:supressLineNumbers/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:sz w:val="24"/><w:i/></w:rPr></w:style><w:style w:styleId="Index" w:type="paragraph"><w:basedOn w:val="Standard"/><w:name w:val="Index"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing/><w:ind/><w:widowControl w:val="off"/><w:supressLineNumbers/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"/></w:style><w:style w:styleId="_5b_No_20_Paragraph_20_Style_5d_" w:type="paragraph"><w:basedOn w:val="default-paragraph-style"/><w:name w:val="_5b_No_20_Paragraph_20_Style_5d_"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:jc w:val="left"/><w:spacing w:line-rule="auto" w:line="288" w:before="0" w:after="0"/><w:ind w:left="0" w:right="0" w:first-line="0"/><w:widowControl w:val="on"/><w:textAlignment w:val="center"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="MinionPro-Regular" w:h-ansi="MinionPro-Regular"/><w:sz w:val="24"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/><w:lang w:val="en-GB"/></w:rPr></w:style><w:style w:styleId="_5b_Basic_20_Paragraph_5d_" w:type="paragraph"><w:basedOn w:val="_5b_No_20_Paragraph_20_Style_5d_"/><w:name w:val="_5b_Basic_20_Paragraph_5d_"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:jc w:val="left"/><w:spacing w:line-rule="auto" w:line="288" w:before="0" w:after="0"/><w:ind w:left="0" w:right="0" w:first-line="0"/><w:widowControl w:val="off"/><w:textAlignment w:val="center"/><w:pBdr/><w:ind/></w:pPr><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Executive-55Reg" w:h-ansi="Executive-55Reg"/><w:sz w:val="24"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/><w:lang w:val="en-GB"/></w:rPr></w:style><w:style w:styleId="Footnote_20_Symbol" w:type="character"><w:name w:val="Footnote_20_Symbol"/></w:style>
<w:style w:styleId="P1" w:type="paragraph"><w:basedOn w:val="_5b_Basic_20_Paragraph_5d_"/><w:name w:val="P1"/><w:hidden w:val="on"/><w:pPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:adjustRightInd w:val="off"/><w:spacing w:line-rule="auto" w:line="288"/><w:ind w:left="0" w:right="0" w:first-line="0"/><w:widowControl w:val="off"/><w:pBdr/><w:ind/></w:pPr></w:style><w:style w:styleId="T1" w:type="character"><w:name w:val="T1"/><w:hidden w:val="on"/><w:rPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:rFonts w:ascii="Executive-55Reg" w:h-ansi="Executive-55Reg"/><w:sz w:val="24"/><w:u w:val="none"/><w:strike/><w:color w:val="auto"/><w:w w:val="100"/><w:em w:val="none"/><w:lang w:val="en-GB"/></w:rPr></w:style>
<w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="Hyperlink"><w:name w:val="Hyperlink"/><w:rsid w:val="006A55B0"/><w:rPr><w:color w:val="000080"/><w:u w:val="single"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="FollowedHyperlink"><w:name w:val="FollowedHyperlink"/><w:rsid w:val="006A55B0"/><w:rPr><w:color w:val="800000"/><w:u w:val="single"/></w:rPr></w:style>

<w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="character" w:styleId="CommentReference"><w:name w:val="annotation reference"/><w:basedOn w:val="DefaultParagraphFont"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:rPr><w:sz w:val="16"/><w:sz-cs w:val="16"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="paragraph" w:styleId="CommentText"><w:name w:val="annotation text"/><w:basedOn w:val="Normal"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:pPr><w:pStyle w:val="CommentText"/></w:pPr><w:rPr><w:sz w:val="20"/><w:sz-cs w:val="20"/></w:rPr></w:style><w:style xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:type="paragraph" w:styleId="CommentSubject"><w:name w:val="annotation subject"/><w:basedOn w:val="CommentText"/><w:next w:val="CommentText"/><w:semiHidden/><w:rsid w:val="007770B7"/><w:pPr><w:pStyle w:val="CommentSubject"/></w:pPr><w:rPr><w:b/><w:b-cs/></w:rPr></w:style>

</w:styles>

<w:docPr xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:displayBackgroundShape/><w:view w:val="print"/><w:zoom w:percent=""/><w:defaultTabStop w:val="720.09"/><w:docVars/></w:docPr>
<w:body><w:p xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:pPr><w:pStyle w:val="P1"/></w:pPr><w:r><w:rPr><w:rStyle w:val="T1"/></w:rPr><w:t>.Theinversestrategyofmaximizingnoisewouldnot only find the way back from IBM to Shannon, it may well be the only way to enter that body of real numbers originallyknownaschaos</w:t></w:r></w:p><w:p xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"><w:pPr><w:pStyle w:val="Standard"/></w:pPr></w:p><w:sectPr><w:type w:val="nextpage"/><w:pgSz xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:w="11907.5672" w:h="16839.9003" w:orient="portrait"/><w:pgMar xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:top="1134" w:bottom="1134" w:left="1134" w:gutter="0" w:right="1134"/><w:pgBorders xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0" w:offset-from="text"/></w:sectPr></w:body>
</w:wordDocument>

Another case of narratives is the narrative that exists outside the software. It lies in its performance. How it acts, where and when. The realisation that people relate to software on an emotional level, makes it possible to create software that tells more than its function. Actually it s possible to tell stories only by how software works. This kind of narrative has been used in some applications of software Art. For example one work which can be found on runme.org. The work is about two Viruses in love. » They search for each other on the net, running through connected computers« (http://runme.org/project/+ViCon/).

This text itself reflects the way how we perceive and talk about software. It shows how unconsciously we use software and the underlying concepts that we touch upon on a daily basis.

I restart Libre Office – this time it works. An empty document opens, and a blinking cursor indicates, that I am ready to type. I switch back to the text viewer and copy the last sentence. After clicking my way back into my editor I paste the string form the clipboard to my empty document. Immediately the text fills the screen: ».Theinversestrategyofmaximizingnoisewouldnot only find the way back from IBM to Shannon, it may well be the only way to enter that body of real numbers originallyknownaschaos«

The polished interface makes us forget about what programmers struggle with every day. The noise that surrounds computation. It is the same noise that should make us aware of how imperfect and subjective software is, but in many cases this noise is being suppressed. Every small glitch is being removed out of software and every irregularity is considered as a bug.
But this noise might instead be the possibility to further explore new opportunities with Code and its execution. Maybe the beauty of software lies in exactly this noise, that is being forgotten about between the logical operations with 0s and 1s.

I save the file and the machine once again writes for me to the hard-drive. I store it using the file-format xml. The file gets stored called NotesOnKittler.xml into the Documents folder. If I open the text in a normal text-editor it turns out, the computer has actually written noise around the actual text that I saved. This noise makes up the standards of the .xml format, encoding information within <tags>.

How to understand software differently.

Surfaces
There are certain trends in software production, that influence our culture towards an increasing gap between sophisticated algorithms and their representation to the user. While the former becomes ever more complicated, the latter is getting more polished with every update. The computational processes get hidden behind user interfaces.
This focus on surfaces, not only reduces software to its interface, but also reflects the current engagement of art with software. The artistic use of machine learning is a great example of the effects of a user that is focused on the interface layer. Instead of engaging with the inner functions of neural networks, artists generate obscure images while mostly talking about datasets, utopia or dystopia. (source) I do not want to say that these approaches are not valid: While it is important to look at the “superficial” layers of algorithms, this should not obscure the underlying technical processes.
What could this artistic practice with software look like? software Art provides an interesting example for an art practice that acknowledges the cultural importance of software at its very core.

Software Art
Software Art describes the »artistic preoccupation with software production« (Cox, 2007, p.147) This means that software Art is using either the software itself or Code as its material. The subject it addresses are mostly the cultural concepts of software (Cramer,

2002b).

The approach of software Art
Software has become so commonplace, that a normal user doesn't even really recognise its existence. A similar effect can be seen in artistic engagement. Software is just part of many digital artworks, not even worth to mention.
Software Art instead does not take software for granted and therefore it also realises how software is made and by whom (Cramer, 2002).
To put focus on the process instead of the end product is not new in the art world, but software Art exemplifies this approach »appropriate to contemporary conditions« (Cox, 2007, p.147). This enables also to think of software in terms of performance. While the result is not necessarily a fixed product, that is visible, it can be a runtime application, that never reaches the state of finishing. An approach like this opens up new discussions and new ideas. An example of this might be the application »Every Icon« by John F Simon Jr. It is a simple 32 by 32 grid that iterates through every possible combination of black and white squares in the grid. The application has been running since January 14, 1997 and will continue for many years. The application only becomes visible, when you visit the website, which displays the current state. Other then that it performs on its own, reaching formations that will never be seen. In a very neat way this work challenges the viewers imagination about limitations of computation, while automatically producing new, unique images.

By taking software as a primary object of study it acknowledges the role of software in a cultural manner, and realizes that software »is not merely a functional tool, but is itself an artistic creation (Nettime.org, 2001). This implies also that the code, which software is made of, becomes the material. It means that software is opened up to much more possibilities. Not only Art will profit from such engagement, but also the culture around software.

Generative Art
Software Art can be seen as a reaction to the narrow use of software in for instance Generative Art. In comparison to software Art the term Generative Art has been around for way longer, following up on Computer Art. But unlike software Art Generative Art doesn't consider software as the primary object of study but uses it, if at all, only as a tool. Furthermore Generative Art is focused on the output (Galanter, 2003).
Going back to the example of machine learning and the current artistic use. The deep dream is not deep indeed. The use of these algorithms is very flat and mostly concentrates only on the output. It's weird morphed images that are being generated on high-resources machines. And they contain for sure very interesting new ways how to program, but this stays untouched by artists. When the images that we see around as outputs of these algorithms can be considered as Generative Art, how could software Art be used to create a deeper understanding of this technology? For example it would also be possible to investigate in algorithms, or part of it or narratives around neural networks itself, instead of showing morphed images that happened to come out of pre-written examples. Of course experimentation with such new algorithms should be welcomed and can be helpful to find ways into new territory, but at the same time it is often being forgotten about engaging with the actual software and algorithm that they are using.
So the artistic engagement with software should not only regard software as a "pragmatic aid" but carefully look at all the different actors at stake (Arns, 2005).

There is a tension between the understanding of Generative Art and software Art that can be productive and helpful to understand new technologies. First of all this distinction makes obvious how versatile software is being used. Secondly this makes obvious the gap between the surface and the underlying material. It is important to talk about both, how software works and how it is represented.

Software Studies
Some past publications have dealt with another examination of software, especially in a cultural framework. Software Studies by Matthew Fuller for instance provides a lexicon with diverse objects of software that are being examined. On the blurb of the dust cov-

er of the book Fuller states: »The growing importance of software makes it necessary to understand [...] the poetics of a loop« (Fuller, 2008). I think this is an important realisation, which opens up the field of software to many different possibilities of understanding and researching.
[especially:
* software Studies, Matthew Fuller
* http://computationalculture.net/ (online journal by Fuller)
* Coding Literacy, Annette Vee
* The Stack, Benjamin H. Bratton
* The stuff of bits, Paul Dourish
* Machine Learners, Adrian Mackenzie
* How to be a geek, Matthew Fuller
* software Theory, Frederica Frabetti
* The Philosophy of software, David M. Berry]

Freeing software
Former artists had to write software to generate, nowadays software is widely available, so it is not necessary to engage with it. This of course means a decline in engagement with software and comes with the risk to take software for granted, without questioning it. But the positive consequence is that this frees programming from certain aspects and gives room for a new engagement, “just as previously the invention of photography perhaps freed painting from figurative representation” (Cox, 2007, p. 155). This also means that software should be used more diverse and could also be abstract. In my opinion software does not always need to have a use, let software be fun. Software should be explored like one plays with photography or different materials of painting. Only that the computer is the stage. Brenda Laurel is writing about Computers as a theatre. (source)

The execution of software can be seen as a performance. When the program is executed the code turns into machine actions.
[elaborate further]

I think that the involvement of art with software can present a useful and contemporary way to change how software is perceived and how we deal with software. The history of software Art shows that this engagement is possible and revealing. In the following I want to point out why I think art can help in the understanding and use of software with a special focus on the underlying processes that are often hidden in programs.

Software is so complex in its relations and so versatile in its effects, that it might be hard to go about a structured analysis. Instead the arts might provide a field of exploration and experimentation, which can at the same time question and enrich the culture around software. Artistic practice has show that it can occupy fields that are not completely understood, like in the field of music. Art offers the opportunity to deeply engage with certain aspects of software and connect the cultural to the scientific realm. Also software can be created by artists to express in new ways and comment on different recent developments. [example]
Although this has to be handled in a subtle way, as a wrong approach can also quickly cause misconceptions. I can cause imagineinations that are not helpful for the engagement or understanding of technology. The potential to buily stories and trigger different imaginations about software or hardware, it a powerful tool to work with as an artist.

»The strong claim for aesthetic computing is that by introducing ideas and methods from art and design into computing, new practices and approaches will emerge responding to new objectives that would not naturally have evolved within the computer sciences and engineering.« (Aestetic Computing, p.31)

The problem of software Archives shows the complexity of software on another layer. Next to his literary work Kittler left a great amount of software as his estate. People archiving his work where confronted with great problems when trying to preserve the software he wrote. (http://traumawien.at/stuff/theory/volume1-feigelfeld.pdf). But how can one archive software? If you only save the program code, this bit of code might very quickly become incomprehensible. Computation changes very fast and so do the programming languages. That means that in a very short amount of time certain languag-

es become deprecated and can not be executed anymore. The most present example is Flash. Many interesting art pieces have been created in this language, but due to many different factors Flash is not used anymore. As a consequence many digital artworks can not be executed easily. So you would have to archive whole frameworks or even the whole hardware with the software? This question challenges many factors and might not be solved in a very long time. It shows once again the complexity and the linkages of software.
A different approach on how to archive could well be thought of through art. In an active way, e.g. if artworks deal with the history and the present of software production, it can be a good way to activate and preserve code and its performance. This can happen through its narratives, through its output or subject.

Conclusion

[–learning about the method I used
– art in software and software in art can be helpful
– towards a more thoughtful use of software]

I close the document viewer and switch back to LibreOffice, I open a new document and it seems like I'm writing, but the computer is writing for me.

References

Arns, I. (2005). "READ_ME , RUN_ME, EXECUTE_ME.'' In: O. Goriunova and A. Shulgin, ed., "read_me: software Art & Cultures", 1st ed. Aarhus University Press.

Berry, D. (2011). The philosophy of software. Basingstoke: Palgrave Macmillan.

Choi, T. (2017). Poetic Computation: Reader. [online] Poeticcomputation.info. Available at: http://poeticcomputation.info/chapters/ch.1/ [Accessed 21 Jan. 2019].

Cramer, F. (2002a). "Contextualising software Art". [pdf] Cramer.pleintekst.nl. Available at: http://cramer.pleintekst.nl/all/concept''notations''Software''art/Software''decontextualizaton.pdf [Accessed 6 Dec. 2018].

Cramer, F. (2002). "Concepts, Notations, software, Art''. [online] Cramer.pleintekst.nl. Available at: http://cramer.pleintekst.nl/all/concept''notations''Software''art/concepts''notations''Software''art.html [Accessed 6 Dec. 2018].

Cramer, F. (2003). ''Exe.cut[up]able statements: the Insistence of Code.'' in Stocker G. & Schöpf C. (eds.), ''Code – The Language of our time'', Linz: Hatje Cantz, pp. 98-103

Constant (2018). The Techno-Galactic Guide to software Observation. Brussels: Constant, Association for Art and Media.

Cox, G. and McLean, A. (2013). "Speaking code". Cambridge, Mass.: The MIT Press.

Cox, G. (2007). "Generator: The Value of software Art". In Rugg, J., & Sedgwick, M. (ed.) Issues in Curating Contemporary Art and Performance. Intellect Books, pp. 147-162.

Finn. (2017). What Algorithms Want. The MIT Press.

Fuller, M. (2008). Software Studies. Cambridge, Massachusetts: The MIT Press.

Galanter, P. (2003). "What is Generative Art?". [pdf] Available at: https://www.philipgalanter.com/downloads/ga2003_paper.pdf [Accessed 6 Dec. 2018].

Goriunova, O. and Shulgin, A. (2004). Read_me. Århus, Denmark: Digital Aesthetics Research Centre, University of Aarhus.

Hadler, F., Haupt, J. and Andrews, T. (2016). "Interface critique". Berlin, Kulturverlag Kadmos Berlin.

Kittler, F. (1992). There Is No software. [online] Monoskop.org. Available at: https://monoskop.org/images/f/f9/Kittler_Friedrich_1992_1997_There_Is_No_Software.pdf [Accessed 11 Feb. 2019].

Knuth, D. (2013). The art of computer programming. Upper Saddle River [etc.]: Addison-Wesley.

Manovich, L. (1999). Database as Symbolic Form. Convergence: The International Journal of Research into New Media Technologies, 5(2), pp.80-99.

Nettime.org. (2001). nettime mailing list. [online] Available at: http://amsterdam.nettime.org/Lists-Archives/rohrpost-0101/msg00039.html [Accessed 11 Feb. 2019].

Reeves, B. and Nass, C. (2003). The media equation. Center for the Study of Language and Information Publication.

Stallman, R. (2010). What Does That Server Really Serve?. [online] Boston Review. Available at: http://bostonreview.net/richard-stallman-free-Software-DRM [Accessed 21 Jan. 2019].

Thought Catalog. (2016). 21 People Share The Greatest software Misuses They've Witnessed. [online] Available at: https://thoughtcatalog.com/michael-koh/2014/01/21-people-share-the-greatest-Software-misuses-theyve-witnessed/ [Accessed 21 Jan. 2019].

Yuill S. (2004). Code Art Brutalism: Low-Level Systems and Simple Programs in Goriunova O. and Shulgin A. (ed.) Read_me: software Art and Cultures, Aarhus:Digital Aesthetics Research Centre.

Software zoo