Tampere University

Mace Ojala

MAINTAIN-ABILITY
A Thesis On Life Alongside Computer Software.

Faculty of Information Technology and Communication Sciences
Master of Science Thesis
February 2022

ABSTRACT

Mace Ojala: Maintain-ability. A Thesis On Life Alongside Computer Software
Master of Science Thesis
Tampere University
Master's Programme in Information Studies
January 2022

Cultural ideas about technology systematically exclude the mundane everyday of maintaining and taking care of them over longue durée. Popular as well as expert views of digital technologies and computer software in particular are oriented towards the novel, new and futuristic. Despite this amnesia, the future is always built on inherited material past, and extends its legacies.

This thesis examines what lessons about living with technology can we learn from software maintainers, who behind the scenes keep necessary digital infrastructures – at least most of the time – in good running order. The empirical material of the research was collected through participant observation and unstructured interviews conducted at four events in Europe and USA. Drawing from science and technology studies and anthropology of technology, I identify themes which concern programmers as they give testimony of their lives lived alongside computer software. My analysis juxtaposes maintenance-oriented programming with maintenance practices in general, and contextualizes biographies of programmers in wider cultural, symbolic and technological infrastructures.

The findings of this thesis challenge the imaginary of existing software as an atemporal object, and complicate the notions of maintenance as low-status work. Software maintainers exercise considerable agency over the immediate material in their care; code. However in doing so, they also find themselves having to articulate dynamic, interdependent and hybrid networks of relations which they are intimately entangled with, and whose durability depends on the success of their ongoing, indeterminate reconfiguration. Both the programmers and the software they maintain must continuously navigate risks of burnout, bugs or falling into obsolescence. Inspired by feminist technoscience and in response to so-called broken world thinking, I theorize the concept of *maintain-ability* and demonstrate its application to foreground the situated, fragile and often underappreciated capacity to give and receive care which holds together more-than-human worlds at the dawn of the third millennium.

Keywords: software maintenance, maintenance, repair, programming, legacy, staying with the trouble

“Programmers at work maintaining a Ruby on Rails application"
Eero Järnefelt, Oil on canvas, 1893

(Collaboration from Jaakko Koskenkorva)

Retrieved from https://classicprogrammerpaintings.com/post/142737403879/programmers-at-work-maintaining-a-ruby-on-rails. Republished with permission.

1 INTRODUCTION

```
From: Kasper Støy <ksty@itu.dk>
To: Mace Ojala <maco@itu.dk>
Date: 2020-11-23 09:00
Subject: Publication script…

Hi Mace,

I noticed that the publication script for taking data out of pure
stopped working… any updates on the way or are we on our own…?

Cheers,
Kasper
```

I received the above email while in the process of writing my Master's Thesis on the topic of software maintenance.

The backstory of the email is that while working as a research assistant in a Horizon 2020 research project called GIFT at the Digital Design Department of IT University of Copenhagen (ITU), I was tasked with designing and setting up several websites for the research projects and research groups at the institution. Some of the websites were straightforward, composed of just a page or two, while others had more elaborate information architecture with a number of areas of the website, and a content strategy to go along with it. Nevertheless, there were a number of similarities across the websites, as all of them included typical elements such as a roster of affiliated research staff, and a list of recent publications – to be expected on sites hosted by and for people in academia. Given the distributed nature of web presence of people and publications across primarily on institution websites, publication venues such as journals, organisational repositories as well as social media and personal websites, the assumption was that updating content on these new, secondary and tertiary websites would be low in anyone's list of priorities. This design issue was captured in semi-formal design briefs as a minimization of content of these websites, and minimization of the labour needed to keep them up to date. The preference was instead to link to more established institutional websites, social media profiles and so forth.

I took it upon myself, anticipating that I might later be tasked the menial annual rounds to survey the affiliated researchers via email for references to their recent publications, that automating parts of the task of collecting references would be wise. I therefore proceeded to program a relatively straightforward plugin for WordPress, a popular web content management system, which would query the institutional publication repository for bibliographical data about publications by the affiliated researchers. Most universities run institutional publication repositories. The faculty research staff is incentivised by the local administration to keep the content updated through sanctioning and disciplinary measures, and less formally by the global research community by expectation to availability of a publication record. Thus, a little PHP program to retrieve this relatively well established bibliographical data from centralised, authoritative repositories and re-using them on these time-limited, auxiliary websites without human intervention seemed like a natural idea to someone like me, employed as an assistant of prestigious and busy professors after an entire career in librarianship and having worked also as a systems librarian. In domain-terms from librarianship and cataloguing, what I was doing was following the footsteps of 19th and 20th century information scientist, documenter and peace activist Paul Otlet in utilising a union catalogue (Wright 2014).

Things ran fine on a number of local websites. While I was not particularly proud of my small, single-purpose plugin, I felt joy as the plugin was found interesting by other peer web-admins at ITU, and I was glad to explain it to them and help them set it up for sites they administered. I was satisfied enough with the plugin to publish it on the public directory of WordPress plugins, thus making a modest contribution to the ecosystem of Free and Open Source Software (FLOSS), the loosely coupled material and ideological infrastructure on which so much of the global digital, and indeed contemporary cultural, social and subjective life depends on and is conditioned on (Kelty 2008).

By the time I received the above email from Kasper Støy, a robotics professor at ITU, a squash aficionado and a supporting paternal figure in the Robotics, Evolution and Art Lab, my research into software maintenance had already led me to attend four events on the topic of maintenance, two of which focused solely on software maintenance. At that time, I considered my current fieldwork complete, and was in the process of data analysis from participant observation and unstructured interviews conducted at these events. My notebooks were a mishmash of first-hand accounts of software maintainers "stories from the trenches" as well as their own (to use a perhaps pejorative term) "folk-theorising" of their own subject-positions in digital infrastructure as I will show in this thesis. I considered myself well familiar with science and technology studies (STS) literature on maintenance and repair, had met some of the prominent researchers in this area, and aligned myself with the programmatic calls to extend science and technology studies and infrastructure studies to attend to the empirical and theoretical questions of maintenance and care (Puig de la Bellacasa 2017; Denis and Pontille 2019; Lindén and Lydahl 2021), not shunning from even the most esoteric New Materialist ethico-onto-epistem-ological theories (Barad 2007; 2014; Tuin 2014; Fox and Alldred 2015).

Despite all of this, Kasper's email admittedly caught me off-guard.

The short message is a crystallisation of the data from my fieldwork, and the theorizations in the published research literature. The first sentence "I noticed that the publication script for taking data out of pure stopped working…" points firstly to epistemic questions of knowing computer systems, and expectations towards them.

In this case one of the websites on which my publication listing widget was installed apparently lacked a listing where one was expected. In this case, the person contacting the author happened to know the infrastructure of the website intimately enough to diagnose the problem, attribute it to the correct “script”, and know which person to contact in case of issues. Infrastructure, as Star famously pointed out, is invisible until breakdown (Star 1999; Denis and Pontille 2019). Rather than providing a comprehensive theory, Star leaves it to ethnography and other empirical research to describe how this visibility is seen – or in Deleuzian terminology how the virtuality of the visible is actualized as the seen (Deleuze 1988). Here, now, was a case in point of that infrastructural inversion (Bowker 1994; Bowker and Star 2000). For how long, and to what effect had the widget stopped working before it was noticed? What happened between Kasper finally noticing it and me receiving his request? The epistemological concerns of uncertainty about whether software is or is not working are a major topic in software maintenance. While fully knowing a system is an ultimately unattainable desire, techniques and tools of monitoring running systems have been developed in response to concerns of “haunted” systems (Kjær, Ojala, and Henriksen 2021). Returning to the first sentence of that email, it additionally points beyond the epistemological to the ontological questions of software. What is software, how do digital objects exist (Hui 2016), what is it for them to “break” or “stop working” and become discontinuous with themselves, and how might we understand the script, WordPress and Pure as distinct objects to name and reference through language? The classical philosophical gap between epistemological and ontological questions (Godfrey-Smith 2003) have been questioned in onto-epistemological work in science and technology studies, in feminist technoscience and in New Materialism(s).

Interpreting the second sentence “any updates on the way or are we on our own…?” we can observe a networked nature of computer software and those whose lives and identities are entangled with it. The antecedent of the disjunctive or-clause presumes maintenance work to be done, while the consequent hints at r disconnection between the plural “we” behind the email, and the implicitly referenced programmer, myself, if the necessary work will not be done. This disconnection is both agential in the sense that whom updating is available to, as well as ethical in the sense that who is left to their own devices if updates are not on the way.

Without going into existential questions of what it might mean to truly “to be on one’s own”, the short email message frames software maintenance as maintenance of socio-technical networks, themselves contingent achievements of articulation and negotiation of these relationships (Latour 2007). Further, the situation foregrounds these relations as vulnerable and fragile, and recognized as such in the two sentences of the short email. The question, thus, is about social-material assemblages which frame the maintainer of software as the primary maintainer of that relation. The second sentence is a clear example of an expectation for “programmed sociality”, the computational conditions of social relations (Bucher 2018).

In response to the email with the above ponderings in the back of my mind, I surfed to the website of the robotics lab, and was confronted with the following error message:

```
Warning: count(): Parameter must be an array or an object that
implements Countable in /var/www/html/wordpress/wp-content/plugins/pure-
feed-widget/PureFeedWidget/Entity/Research.php on line 101
```

Error messages are key in programming, and give valuable insight into the operation of the code. But this code did not look familiar to me. It seemed to have a more elaborate object-oriented design than I knew I had time for when programming the widget. Upon closer investigation with the IT department which operates the underlying platform, the WordPress instance, the running widget was found to indicate itself to be version number 0.3.0. This was surprising, as the latest one I have published is 0.2.0 – the code reported to be broken was not of my making. Instead it was a "hard fork" ie. a derivative program based on a copy of the program, with significant redesign and without coordination or merge with the upstream developer, myself (Zhou, Vasilescu, and Kästner 2020). How was I to engage with dealing with the error, and repairing the entanglements of research labs, the institutional repository, servers, labour relations, academic citation networks, Kasper, moral commitments, and the digital infrastructure on which they (although in a modest way) had come to depend?

This story highlights many of the issues and theorizations relevant for software maintenance, and thus serves well as an introduction to this thesis. As we will come to see as this thesis proceeds, contrary to widely held *tabula rasa* based views, much of programming takes place *in media res*, in the middle of things – in the thick of it, and as matter comes to matter, to use New Materialist phrase from Karen Barad (2003). Critical software studies scholars have further worked to deconstruct the ideology that software is strictly determined by its source code (Marino 2012; 2020). This work has focused on performativity and execution of code (Chun 2008). Similar line of reasoning is implied by the contemporary critiques of algorithms on social media, in predictive policing, advertisement, credit scoring et cetera, when their fairness, explainability and accountability (or rather lack of) is questioned from social justice points of view (Lee and Björklund Larsen 2019). These critiques have focused particularly on algorithms derived through machine learning techniques from historical data (O’Neil 2016; Noble 2018), and whose operation is not straightforwardly attributable to their creators, whether these are understood to be *auteur* programmers, or the companies employing them. These algorithms are not defined by their source code, but rather they enter an already busy world of pre-existing training data on one hand, and the context of use in the present on the other (Mackenzie 2006; 2017), causing all kinds of unexpected consequences. And like it is the case for this minuscule selection of high stakes algorithms, so it is for all software: it is not reducible to some original intent by a willing, fantasmagorical Subject projecting itself through the logical consequences of the text of source code (whether programmed or machine learned) onto the external world. Instead it is worthwhile and interesting to consider the much more diverse networks of causality crisscrossing backwards, sideways and in loops across human and non-human actors as they (we) stumble through a vulnerable existence in time and space in which they find themselves (Mackenzie 2006), just like that WordPress plugin does.

In summary, even at its conception software arrives *in media res*, in the middle of things. Rather than accepting that something is to be carved on a *tabula rasa*, constructed *ex nihilo*, we will do well by paying attention to the relational ontology and genesis of software – how it comes to be, and remains in being. Programming is always improvisation, and its result a *bricolage* of elements chosen from what is available, arranged next to one another. New relations are folded from what already exists (Cohn 2019; Bialski 2020) in a dynamic and active environment, symmetrically setting the future technological *a-priori* for the activities yet to come (Kittler 1992; Krämer 2006; Tuschling 2016). Rather than universalizing or totalizing software as an abstract unity, a lens of situatedness from feminist technoscience studies is useful here (Haraway 1988; Suchman 2012b); configurations are always re-configurations (Suchman 2012b; see also Latour 2012). Software is a weaving and re-weaving of what already is and already was.

Since its beginning in the late 1980s, science and technology studies (STS; also as science, technology and society) theorised technology not as a growing list of achievements, but as an ongoing process of social construction of technology (SCOT) (Pinch and Bijker 1984). Constructedness has been a key theme in the literature since, and technology and society are seen to continuously co-construct one another over time. While the lens of maintenance has not always been a primary focus in study of technology, especially with the feminist post-Actor Network Theory corrections to the field of STS have developed the questions of who gets to do what work, at what cost, and for whose benefit (Denis and Pontille 2019). Particularly the conception of *care* has developed these lines of research (Mol, Moser, and Pols 2010; Puig de la Bellacasa 2011; 2017; Lindén and Lydahl 2021).

While whether one particular item or technology gets maintained over time is perhaps interesting, the stakes of maintenance extend well beyond fixing machines and tools – what we might consider repair (Graham and Thrift 2007). Maintenance on the other hand, is about conditions of possibility of cultural, ethical, political, social, psychological, economical and creative life not only in the futures, but importantly as experienced and lived in the now.

Digital technology has become a visible object of desire, anxiety and optimism in various discussions even in the mainstream for example about education, job prospects, and what is “cool”. Besides all the new and exciting things digital technology is or is promised to be, it has also gradually since the second World War become a key background factor, both materially and discursively. What’s above or beyond the horizon of *technological imaginary* (Balsamo 2011) has been controlled by a narrow set of technologists and their allies, such as those with agenda-setting, communicative and of course economic power. Technology which used to be exciting, if socially constructed to be successful in the long run, has receded into the background – we call this “infrastructure” (Star 1999). Some theorists argue, in alignment with Marx’ dialectical materialism, that only when technology becomes backgrounded and “mundane”, it comes to matter most (Chun 2016; Vinsel and Russell 2020). We might call this opposite of technological imaginary technological the ordinary or the technological real. The ordinary and real infrastructure becomes visible on breakdown (Heidegger 1977; Star 1999). But it is worth asking exactly what structures and activities keep the infrastructures, most of the time, from breaking. What processes and what work keeps most of software out of sight and out of the realm of the imaginary? How is software practically suppressed into the infrastructure? And further, who does this work, at what cost, for whose benefit at a particular time in a particular place?

Various theoretical views attempt to reduce software to something else, for instance to electrical components (Kittler 1995), services (Kaldrak and Leeker 2015), mathematical logic, the intent of the original designer, social structures, ignorant or arrogant neo-liberal ideology projected from Silicon Valley, capitalist domination and exploitation, or even to the malign. In the relationist research traditions we reject single reductions to what “things really are”, and instead accept multiple versions (Mol 2003). Methodologically we choose to “follow the actors” in an attempt to provide a plurality of partial views rather than a God’s eye tricks of a singular, total view (Haraway 1988; Benzer and Reed 2021). Besides more-or-less distanced academics, practitioners themselves such as programmers have no shortage of understandings of the situations in which digital infrastructures persist, and how they are entirely codependent with “non-technical” concerns. Of course no universal consensus exists among practitioners either. And while there is no need to take the “indigenous” knowledge of practitioners as a final authority, it ought to be taken seriously (Callon 1984).

Methodologically, we must not limit our analysis *a priori* to certain types of objects and at the beginning of our study assume that there are technological realities and social realities (Callon 1984; Latour 2007). We must be wary of what ontologies we accept, and how they are carved. Post-modern, post-structuralist and deconstructionist theoretical groundwork help us reconsider continuities across familiar and problematic ontological gaps and conceptual entrenchments such as nature/culture, science/technology, material/discursive, individual/collective, history/future and fact/artefact (Haraway 1988; Benzer and Reed 2021). Such divisions are the *explanandum*, not the *explanans* of theory in classical Actor-Network Theory tradition in science and technology studies which insist on *flat ontologies* (Latour 2007).

Thus the weavings of software must be allowed to include (m)any kinds of things, and importantly non-code things. Software is thus bound in material causes and effects well beyond its own ontological categories of software (itself a historical construction like any concept (Haigh 2009; Ensmenger 2010a; Marino 2020), or its quintessential material, source code. And is this reach beyond not exactly the promise of software? Not that digital technology is an abstract exercise, but that something can be achieved with it.

One particularly fruitful lens to study the richness of a technology is its maintenance. By shifting our focus onto maintenance, we necessarily come to a thicker description and richer appreciation of the materiality, practises and relations in which it participates. Study of maintenance has been explicitly called for by scholars such as Susan Leigh Star (1999; Bowker and Star 1999), Stephen Graham and Nigel Thrift (2007), Jêróme Denis and David Pontille (2015; 2019; Denis, Mongili, and Pontille 2016) and Steven Jackson in his invitation for *broken world thinking* (2014). Empirical research literature covers many areas of technology maintenance from cars (Henke 1999) to mobile phones (Nova and Bloch 2020) to data (Pink et al. 2018). Feminist scholars have developed maintenance toward the concept of *care* both as an ethic (Tronto 1993) and as an ontology (Puig de la Bellacasa 2017). Maintenance and care feature powerfully and beautifully in Marisa Leavitt Cohn's ethnographic work among NASA engineers and scientists in the Cassini mission (2014; 2016; 2019; 2021), a key inspiration for my own research.

This thesis is my own inquiry to how theorizations about maintenance in general play out in connection with software, an interesting kind of material so integral to the digital world already since the mid 20th century. Rather than wasting time in a futile attempt to reveal the final truth, I've tasked myself to investigate, listen in and describe some of the issues of maintaining software as raised by practitioners, and learn and theorise about life alongside computer software.

This text is first and foremost my Master's Thesis in which my task is to rehearse and demonstrate the will and capacity to conduct a research project well. Secondly this is an analysis of software maintenance, synthesising on one hand my qualitative fieldwork and on the other the published research literature from science and technology studies on the topics of repair and maintenance. Thirdly this thesis serves as a sounding board for reflecting on the first-hand experience of inhabiting and coping-with a technological *a-priori* (Kittler 1992; Krämer 2006; Tuschling 2016) into which we have been *thrown* (Heidegger 1996), and in which we find ourselves, together. To limit the scope of my work, I focus on aspects which regard computer software, and the ongoing maintenance of the entanglements it is part of. In my writing I aspire for an ethnomethodological style of description, and at this stage of my research into digital culture I take a non-moralist stance. My aim is to contribute to software studies and academic debates about digital technology by nuancing, de-essentializing and diplomacy. I hope to describe what this ongoing material encounter, software maintenance, means to relevant entanglements of people and material. This is my overarching research project. I further explicate this project by deriving from literature a handful of specific research questions for this thesis below. As a contribution in the present and hopefully in future work too, I hope to enrich what could possibly be learned towards maintenance of a material, broken world. To do this, I listen to the voices of those dealing practically with existing software, and give a voice to often disregarded software which already exists. These are my informants, and this work is through-and-through indebted to them.

The work proceeds as follows: after this introduction and a brief personal positioning, a review of relevant literature especially in science and technology studies is given, and my selection of method is argued for. After that my fieldwork at four maintainers events to share and discuss experiences and concerns is described. I will then analyse and systematise my observations and compare the two software related events to the two generic maintenance events. I will then proceed to discuss how the software maintenance events compare with published literature on maintenance, and provide some critique and develop new theory. I conclude with some reflections on the thesis project, and sketch some future directions of research based on this work.

1.1 A preflection: writing from somewhere

Before proceeding into my analysis through a literature review, I wish to include a brief positioning of myself as an author, as a researcher and as student as well as a programmer, in response to critiques of erasure of the researcher of assumed neutrality and the “view from nowhere”, which Donna Haraway has labelled as one of the “God’s eye tricks” (Haraway 1988), and as done in the ethnographical anthropological literature, an exemplar being Sharon Traweek’s self-exposure in her *Beamtimes and Lifetimes* in which she positions herself onto the fieldsite, and makes visible her position in the knowledge production about high energy physics (1992). The aim on this brief section is not self-centering, but rather a feminist reminder of the impossibility of self-removal, and the gap of subject-object dichotomy. I also need to write this section for my own sake.

I am writing this as my Master’s thesis, overdue by a few years in the standard Bologna system of a 3 years Bachelor’s degree plus a 2 year Master’s degree. I graduated my Bachelor’s degree, started already in 2013, only quite recently in late 2019. I am writing this thesis for Tampere University remotely from Copenhagen where I moved to already two years after admission into my combined Bachelor and Master’s, as I found both great joy in and highly relevant employment for my academic pursuits after an Erasmus exchange at University of Copenhagen. Since then I have visited my *alma mater* in Tampere only twice, each time for less than a day. A new logo, a new strategy and new organisational structure of Tampere University have been instantiated, and I’ve lost track. To be quite frank I am not anymore quite sure what my programme is called, which faculty my academic seniors are employed at, and where do I belong in the organisational chart. My home has instead come to be – for the time being – between the ETHOS Lab and Technologies in Practice research group on the one hand, and the Digital Design Department at IT University of Copenhagen. In this life-situation I consider myself a fortunate beneficiary of the global corona crisis which is ravaging the planet in that the Master’s thesis seminars at Tampere University are offered virtually as the world is ebbing and flowing in and out of partial lockdown. The *suurpiirteisyys*, perhaps best translated as generosity, of the academic and administrative staff at Tampere University needs to be acknowledged, as without their supportive flexibility my academic career (“queereer”) so far would have been much impoverished.

While I enjoy considering my fledgling academic career *a queereer*, I recognize firstly that I am anything but alone as an academic in having a non-standard path, and secondly that having the opportunity to have substantial agency in carving non-standard journey is a privilege and hinges on agency of others. Everyone’s path in life in general is unique, and while some structures such as an University education give the paths embarked on some common shape (see some critiques of industrial and commodified nature of contemporary University education in (P. J. D. Gielen and de Bruyne 2012), everyone navigates through them on their own, and alone. The capacities for “free-styling” are, of course, a function not only of one’s character, ambitions and dispositions, but of one's class, economic, social and cultural capital (Bourdieu 1986), gender, race and where in material history one happens to be thrown into. I join a lineage of white men, dead and alive, all of whom I am grateful to too. Many hands have nudged my journey, as theirs was nudged too. My peer Aleksi Ruuhilahti referenced me to Taina Bucher’s awarded PhD dissertation *Programmed sociality: A software studies perspective on social networking sites* (Bucher 2012) which was undoubtedly the single most important individual reference I’ve ever been gifted with. It got me into Software Studies. Minna Saariketo backgrounded academic hierarchies in favour of theory and staff-student encounters. Saskia Scheltjens’ invitation to Ghent Center for Digital Humanities uprooted me, and allowed me to work alongside generous Sally Chambers. Laura Beloff took my hand and introduced me to Anders Sundnes Løvlie, securing my professional involvement in academic life in a H2020 research project for a number of years. My thesis supervisor Tuomas Harviainen made themselves available more than expected and caressed my ego productively and professionally. The patience of Silja Jäntti at the study services of my faculty is remarkable. Finally Marisa Cohn gave me legacies, maintenance and obsolescence of computer software as a topic, her hospitality an opportunity to work myself into them, and her friendship has carried me forward. Thank you.

Finally happenstance and fortune always play a role in human conduct – an existential position I hold dear. I believe agency of my own and of others (human or non-human) in this unlikely and absurd journey of transformation is partial.

What I have set out to do with this thesis is of course to graduate from the Master’s programme in which I am enrolled in, by hoping to convincingly demonstrate to my senior peers and supervisors sufficient competency in conducting academic research. Further, I am using this opportunity to collect, collate and come to terms with some of my own personal experiences and reflections about life growing up alongside software as a hobbyist programmer at the turn of the third millennium, in my prior librarian career in heritage institutions, and my current one in academia trying to cherish a critical technical practice (Agre 1997; Stevenson and Helmond 2020; Rieder 2020) by engaging in programming. Finally I try to lay out some future paths for myself which I hope to pursue in a future PhD and beyond. Part of this thesis is my attempt to do anthropological and philosophical groundwork for myself for thinking about software in general, and software maintenance in particular. I hope this work prepares me to continue on this path.

I remain opaque to myself (Butler 2001). And thus this brief section too remains an impartial, but sincere, positioning of myself in my research project and in my thesis. For the entirety of the thesis, I will try to keep my voice present when appropriate.

2 CONCEPTUAL FRAMEWORK AND DEVELOPMENT OF RESEARCH QUESTIONS

What does it take for something as supposedly ephemeral as computer software to endure over time? How does it acquire and retain its seemingly surprising capacity to endure (Ensmenger 2014; Kelty and Erickson 2015)? What does sameness-with-oneself, *identity* mean for software, and what might be some conditions and implications of this persistence if, as some argue, software infrastructures contemporary life everywhere? These are questions I have myself pondered, and as we come to see these are also questions which trouble fellow scholars and my informants.

A first response might be that as a logico-mathematical, abstract object software does not exist in time and questions of its persistence are hence irrelevant. While strands of theoretical computer sciences do treat software formally, the underlying theory of science assumes the ontological existence and epistemological access to transcendental universals (Godfrey-Smith 2003). Ontological and epistemological positions of such a theory of science were questioned during the 20th century by social constructivists (Godfrey-Smith 2003), who described how scientists transcribe their practical work onto documents and other artefacts. Science was thus framed as a literary practice and a social construction (Latour 1987). Beyond science, technologies of the literary have more generally been studied by media scholars such as McLuhan (McLuhan 1962), Kittler (Kittler 1992; 1999), Derrida (Derrida 2005), and Landow (Landow 2006), Ernst and Parikka (Ernst and Parikka 2012) and Kirschenbaum (Kirschenbaum 2016) just to mention a few canonical men. As this thesis does not dwell on questions regarding abstract universals, or seek to reveal a Philosophical mode of existence of computer software (the interested reader is pointed toward (Hui 2016)), to bracket out the line of reasoning of software as transcendental it suffices to say that the discursive, including theoretical computer science, has a constructed technological basis.

A second, a commonsensical response to the conundrum about persistence of software might be that software remains the same simply because it was not changed: things that have not been changed are the same as they were. Newton’s first law, the law of inertia (Newton 1687) applies. A sufficient answer; end of story! However in their chapter *Why do maintenance and repair matter?* Denis and Pontille (Denis and Pontille 2019) suggestively give the label “maintenance and repair studies” to a growing body of research literature which both empirically and theoretically engages with the necessary practical work of making the material world last. Synthesising published literature since the 1990s, they argue that foregrounding the often backgrounded maintenance and repair of technology can firstly unsettle and nuance the well-studied breakdown events, and secondly bring more depth to theorization of materiality itself.

It is not a radical suggestion to understand software as a technology. As a technology, ie. as an interested, material and practical application of a system of scientific ideas, software's persistence hinges on its materiality and the practises it participates in. The material of software is source code – a common-sense notion also put productively to analytical use to trace the performance, circulation, changes and solidification of software (Mackenzie 2006). The name given to the quintessential practice of production of this technology is "programming". These two popular but abstract concepts of "code" and "programming" will come to be rethought in the course of this thesis.

The literature about repair and maintenance Denis and Pontille summarise pivots on Susan Leigh Star's influential *The Ethnography of Infrastructure* (Star 1999), a paper published in parallel with the book *Sorting Things Out* which Star co-authored with Geoff Bowker (2000), and which is canonical reading in both Science and Technology Studies (STS) as well as in Information Studies (IS, or Library and Information Studies alias LIS). On a personal note, my academic journey pivots on this book. Both of Star's works make a similar argument, and call for study of the "boring" such as standards, classification systems, protocols, and in general what is considered infrastructure as worthwhile because they structure much of the world which we inhabit. Ethnographers have earlier demonstrated the usefulness and interestingness of *thick descriptions* (Geertz 1973) of the mundane, the everyday and the-taken-for-granted, but Star points the ethnographers attention specifically to technological infrastructure, and the lives lived alongside it.

To add to the study of everyday technological infrastructure, turning it almost to an imperative, Wendy Hui Kyong Chun argues in her provocatively titled book *Updating to Remain the Same* that media technologies gain their importance when their use becomes habitual rather than when they are new and novel (2016). While new technologies capture much of the technological imagination (Balsamo 2011), attention and excitement, pumped up by innovation-speak, the ones which get to be maintained and exist over the longue durée are what matter most (Vinsel and Russell 2020). Boring, mundane and "backgrounded" technology sets the scene on which the everyday performed. Researchers have, of course, not been immune to the lure of the shiny and the soon-to-come.

Recently, critique of classification and its consequences, so powerfully done already by Bowker & Star (Bowker and Star 1999) has gained new interest with the mainstream extensions of computational culture. Particularly the excitement, promises and imaginaries around machine learning (ML) and artificial intelligence (AI) have set the research agenda. While AI is, perhaps, is more accurately seen as automation rather than "intelligence" as argued by Manovich in a critique of AI (2015), classification ie. assignment of data points to predefined categories is indeed a key task of ML-based artificial intelligence – the other being clustering ie. grouping together data points by their proximity and inferring new patterns (James et al. 2017). The core social issue is the automation of classification of people. Critical research has demonstrated that when algorithms are derived from biased data through the techniques of machine learning, the outcome algorithms will reproduce biases in the training date. Socially problematic biases are those based on protected attributes such as gender and race, but also of class, criminal status, religion, health, consumer potency &c. Machine learning algorithms rely on large quantities of training data, and this data is misrepresentative, ie. biased, due to how data collection, archiving and access has historically been organised.

Systems derived through machine learning are not explicitly programmed in the traditional sense of programming. The lack of a programmer leaves an agential void, and thus an ethical void. Further, how these systems work is hard or perhaps impossible to understand after they have been constructed, and their behaviour is unavailable for prediction or explanation. Many socio-politically relevant and consequential algorithms are buried deep in media technology platforms operated by megacorporations who extract value from feeding and operating these algorithms (O’Neil 2016; Noble 2018; Zuboff 2019).

While framing The Algorithm as a character in *algorithmic drama* (Ziewitz 2016) is simplistic, it productively allows critical, evaluative discussion of the role of algorithms proper place and conduct to be had both in academia as well as in public life more widely. The risks are real and some of the consequences of “bad algos” widely documented (O’Neil 2016; Noble 2018).

Ironically, debate and critique of high-stakes algorithms sides with the “winners”, exactly as critiqued by Star, according to Denis and Pontille (Star 1999; Denis and Pontille 2019). While interesting and important, targeting individual algorithms produces a very lopsided critique which is basically moralist. It tends to promote an imperative that it is the FAGA (Facebook, Amazon, Google, Apple), FANG (Facebook, Apple, Netflix, Google) or whichever US American megacorps happen to be popular at a time we as researchers and as as societies ought to be most engaged with. While this important, *exposé* work must be appreciated, it is unwilling and unable to account for very much beyond its object of critique. From information studies, media studies, cultural theory, anthropological or philosophical perspectives an *exposé* has little to contribute to theory, ie. questions of *why*, beyond providing a data point. A toxic problem now appears: if “artifacts have politics” (Winner 1980) and only morally corrupt cases are shown, this implies that technology is harmful, and perhaps even worse, that politics is harmful. No room is left in critical research for cases where technology did good, or where it was insignificant. By its shape, these arguments necessarily highlight the powerful and hegemonic, and in practice fixate the inquiring gaze on the usual suspects, a handful of USAian tech megacorps while leaving little to say about everyone else, effectively marginalising almost everyone. This is surely not the intent of any work in the critical theory tradition (Horkheimer 1972). Rather this conclusion is an epiphenomenon of the unfortunate shape of the arguments skewed towards hyperbole, I am convinced an unintended aggregate effect may be worth addressing in a review research on it’s own.

Rather than software algorithms as the characters of a drama (Ziewitz 2016) or a yet another ‘software crisis’ (Bialski 2020), a different view of the performativity of software is available if we think of it more ethnographically as infrastructure, following Star (1999), focusing on its ongoing repair and maintenance (Graham and Thrift 2007; Denis and Pontille 2019). We can get beyond the too abstract, alienating and masking notions of “algorithm”, “data”, “software” and “programming” to the material of code and to the lived lives of the people who participate in it (Bialski 2020).

Rather than a set of objects, software is *multistable*. The notion of multistability from post-phenomenological literature gives technology multiple, competing and temporarily stable shapes and meanings, rather than finality (Ihde 1993; Verbeek 2001). What at first seems like the “same” software is experienced in multiple ways. The word processor I am writing this on, for instance, is experienced very differently by me as I struggle to type this thesis versus a programmer who is concerned that its code is in working order. The maintainer might also experience this software as typists themselves when they sit down to write, analogous to Star’s literary character of the plumber who pauses their work of fixing pipe infrastructure to pour a glass of water to quench their thirst (Star 1999). As Mackenzie puts it about software, originators and recipients of software oscillate between these positions, sometimes using code and sometimes producing it (Mackenzie 2006). This oscillation of positions comes across in the voices of software maintainers, as we will encounter below in my analysis. An analogous concept in anthropological research literature, developed in the medical and embodied research context, is Annemarie Mol’s *version* (2003). My the typist’s version, the maintainer’s version and all the other versions together constitute the object known as this word processor. We do not need truth about software, we need it to be workable. Here the pragmatic philosopher (the “pragmatic maxim” in Peirce 1935) and the pragmatic programmer (Hunt and Thomas 1999) find a common ground.

In summary, objects, artefacts, technologies are not final. Rather, the ongoing negotiations about their interpretation and use grant them identity and persistence, and on which they hinge. The negotiations do not exist exclusively or primarily in the linguistic, representational or abstract social sphere as the notion of “negotiation” might suggest, but, as post-structuralist theorists have corrected the posits of their predecessors, are deeply rooted in the material and the situated. The theoretical concepts of “sociomaterial”, or “sociotechnical” are awkward and do not do away with the dichotomies they bridge, but are nevertheless worthwhile for communicating the gist of social construction of technology (SCOT). Early theorizations of social construction of technology were advanced among others by Pinch and Bijker (1984). Influential empirical studies of negotiation of technology include the embedded ethnographies among photocopier engineers by Suchman (Suchman 2012b), and inside a information system development project by Star and Ruhleder (Star and Ruhleder 1996). Integrative application of similar ideas can be found throughout the wide field of Human-Computer Interaction (HCI), information studies (Ingwersen and Järvelin 2005), and the design maxim “user completes the design”.

Pushing it further than a “new media”, Manovich has called computer software *meta-media*, a media which can sufficiently incorporate other media formats onto itself through emulation (Manovich 2013). After the emulation is sufficient, other media formats seen through this meta-media can and do develop in unique ways. This media-theoretical analysis aligns with basic theory of computer science, which posits a Turing-complete machine is equivalent with any machine (Sipser 2012). Examples include the world wide web as more than a collection of documents on screen, or

Photoshop as more than photo manipulation on screen, or a computer game as more than a boardgame on screen. It is programmability which makes one computational machine theoretically into any other possible computational machine.

How this theoretical potential has been actualized has been studied by media archaeologists who have put hermeneutic techniques to use to interpret historical media machines and apparatus such as audio compression algorithms (Sterne 2012), digital images and most relevant for the purposes of this thesis, computer code (Manovich 2013). These critical interpretations have shown how human life not only leaves its mark on these artefacts, but is lived through them and how their multiple meanings remain open rather than closed.

Further theorizations of the ongoing entanglements of objects and artefacts and stronger inclusion of the material, ie. beyond the discursive, have insisted on inclusion of the “non-human” at the very core of analysis. Various so-called New Materialist frameworks (Fox and Alldred 2015) explain how “things come to matter” (Barad 2003; 2007). An early, perhaps the most influential such contemporary theory was Actor-Network Theory often attributed to Bruno Latour. ANT continues the deconstructionist task of undoing dichotomies, particularly those of the social and technical, and human and non-human. As one of the “flat ontologies”, ANT refuses *a priori* ontological categories such as the social and technical (Latour 2007), and rather insists the categories to be the *explanandum* of research of socio-technical assemblages. The ongoing push and pull of *translation* of action or agency, a central concept of sociology, between various kinds of actors is what allows heterogeneous networks to exist, and endure over time. ANT has investigated and shown how technology is society made durable (Latour 1990). Note the active mode of the verb “made”.

Deconstruction of received ontological systems, and inquiry into the structures and processes of which they are outcomes of, aligns well with the longer critical theory traditions. The social construction of technology theorizations long been in close and extremely fruitful interaction with critical feminist scholarship. Feminist scholarship has drawn from, amended, enriched and corrected study of science and technology particularly by "bringing back the body" and making room for silenced voices, including those of non-humans, but also oppressed humans such as women. Monumental works of scholars such as Donna Haraway’s (1985; 1988; 1997; 2016), and Karen Barad’s (2007) have brought relational ethics and care ethics (Tronto 1993; Collins 2017) front and centre to the study of technoscience, and are asking very demanding questions about the art of living and dying on a damaged planet (Tsing 2015; Haraway 2016).

I circumscribe a theoretical research framework for analysis of software from three standard foundations of contemporary, pragmatist Science and Technology Studies. These foundations are materiality, practises and relationality.

2.1 On materiality

Firstly, to analyse materiality is to see outside and beyond merely of the discursive. While the Western philosophical literature on materiality goes deep, in social or cultural studies to attend to materiality is to resist focusing only on representations (in language or signs more generally, after the "crisis of representation") or abstraction to the likes of the notions of "technology" or "the digital". Computer science theorists tend to frame the central questions of computing to be about manipulation of formal symbolic systems in the mathematico-logical tradition, while social science theorists tend to reduce software to their own familiar objects of study such as social relations, power and domination, agency, or identities such as class, gender and race thus removing software from analysis. (Mackenzie 2006; for a critique of traditional conceptualization social relations see e.g. Latour 2007) . Research is always a balancing act and question of choice, and by glossing over certain details, both immaterial reductions of the computer scientists and the social scientists enable some analysis, disable others. Attending to the concrete properties of material is a useful counterbalance to social constructionisms of various flavours when studying meanings of technologies, and media technologies in particular (Lievrouw 2014).

Keeping code close to the centre of analysis grounds study of software as material. While code is mutable and malleable, it is also hard to understand, circulate and modify – as material of software code resists. Exactly the situations where it finds temporary stasis or change are interesting (Mackenzie 2006). Maintenance of software points to fragility as material property.

Attention to materiality also keeps the bodies of programmers within the analysis, and rejects software as a cognitive operation, "a view from nowhere"; like all knowledge, knowing code is situated knowledge (Haraway 1988; Suchman 2002): code is intimate, code can be felt, code can damage and it can bring joy. Code can stay when its maintainers leave or die. Material is continuous, and by following bodies of programmers my fieldwork brings me (=my body) into proximity with bodies of code.

2.2 On practices

The second element of my theoretical framework are practices. To attend to practices is to attend to the “on-goings” of the everyday. So-called practice theory involves attending to the mundane and uninteresting, the routine, and the insignificant (Reckwitz 2002). Most of everything is uninteresting, at least to those already involved in it. It follows that study of everyday practices is the most important, and this is the research programme advanced by cultural anthropologists, ethnographers and ethnomethodologists.

Beyond describing and cataloguing them, analysis of practices attends to how some activities become practices, for whom, what their limits are and how practices change and are negotiated and how some activities receive their invisibility as mundane "on-goings". While social theory often attends to practices only theoretically, the roots and commitments to taking practice seriously lay in Pragmatist philosophy (Bogusz 2018). To take practice seriously, “experimentalist” social theory of practice ought to see social theory of practice continuous with practice itself rather than separate from it. One of implication is for the experience of the research subjects and the researcher to to be seen as continuous with one another. (Bogusz 2018). The researcher is thus never separate from the field and the informants.

Many fields of research have taken a “practice turn” around the late 20th century. Relevant for this thesis, the primary role of practices as the research object in contemporary ethnographic anthropology has radiated to specific domains such as system development (Star and Ruhleder 1996; Suchman 2012b) and information behaviour (Nyce and Talja 2015). As an extension and correction of sociology and anthropology, science and technology studies has since Actor-Network Theory complemented the enquiry of (arguably misleadingly anthropocentric and abstracted) social practices, and detailed and theorised how machines, natural objects, scientific instruments, inherited architecture, infrastructures and in general “the non-humans” participate in and constitute practices and in the mundane everyday. This extension of practice theories thereby re-grounds practices in the material world.

Cultural anthropologists have developed qualitative research techniques to interest themselves in the everyday practices, and allow their fieldwork to surprise them (Winthereik 2019). One of the techniques is to attend to breakdowns: when things break down the familiar becomes strange, and what was backgrounded is foregrounded. Garfinkels's sociopsychological "breaching experiments", disruptions to the normal flow of the social everyday, are a (controversial) way to operationalize study of the everyday routines (1967). Breakdowns of technology foreground how practices are through-and-through imbued with the technological. The canonical example from phenomenology is the thought experiment of a hammer given by Heidegger (1977), as the tool jumps from background to the foreground when it breaks. Fortunately digital technology such as software offers no shortage of breakdowns as opportunities for researchers to draw from as material for thick descriptions (Geertz 1973) of social and cultural practices.

2.3 On relationality

The third and final cornerstone of my overall conceptual framework is relationality. To study relations is to study connections and disconnections between entities, or in the strong programmes of relationalism, the relations are not only primary, but strictly constitutive and productive. Regardless of the specific flavour of relationalism, the central object of research is relations. Relational ontologies are non-essentialist or anti-essentialist and focus on co-production, co-constitution and interdependence. They are philosophies of becoming rather than philosophies of being. Relationist philosophies have been greatly advanced by feminist scholars, as well as those of science and technology with very productive exchange in between (Puig de la Bellacasa 2017). Networks of relations are dynamic, always potentially changing and never evenly distributed. Relations are reciprocal, and there is no outside of relations.

Research which builds on relationalist ontology analyses how things hang together, rather than exist apart or in isolation.

The roots of science and technology studies are relationalist. The (in)famous, and controversial Actor-Network Theory (ANT) often attributed to Bruno Latour emerged as a strong stand against the theory of social construction of technology (SCOT), itself a reaction to technological determinism (Pinch and Bijker 1984). ANT argued for agency of the material and artefacts. It's innovation was to defuse the arguably bankrupt argument between technological and social determinism by insisting on "a flat ontology" of network of heterogeneous actors, an ontology which would entirely deny the assumed dichotomy of social and technical (Latour 2007). ANT was by no means a first Philosophy, or rather a method as insisted by one of its founding figures (Latour 2007), attributing agency to non-humans, but it brought it back into social sciences from which it had been purged. This deconstructionist move avoided the entrenched positions between technological and social determinisms, and opened a refreshing field for new empirical work for description and analysis of, among other things, how the categories "the technological" and "the social" were materially achieved in practice as the *explanandum*. Early, "radically empirical" ANT research started as an extension of the sociology of scientific knowledge (SSK) in laboratory studies (Latour and Woolgar 1979), and later extended to study of technoscience (Latour 1987) and technology and material culture in general.

The work is not to verify or test preconceived and received theoretical concepts for their validity, but to use them generatively and productively. By taking a methodological "scrupulous detour through the empirical" (Latour 2007; Ang 2011; Winthereik 2019), research tradition of STS often involves ethnographic approaches. When heading out to the field, the researcher considers theories and concepts as nothing more than prototypes which interact with the world. They are provisional and suggestive, sensitising and productive. (Winthereik 2019). Successful fieldwork provides surprise and elicits relationality (Winthereik 2019). The canonical "scallops paper" (Callon 1984) included *Pecten maximus* scallops of St. Brieuc Bay as equal peers to scientists and local fishermen in a network of translation. The Zimbabwe bush pump at the same time constitutes and is constituted by a complex network of relations in another work (de Laet and Mol 2000), and we find another pump in post-purchase study of air pumps describes how gender and domestic relations are maintained in maintenance situations (de Wilde 2021). Description of technology as relations, and the shifting, stable and fragile variants of its objects (Denis and Pontille 2015) are the staple of science and technology studies.

2.4 Living as well as possible in a broken world

What do we see when we look at computer software through the perspectives of materiality, practices and relationality as they are described in this thesis? Can we gain insight into how software comes to matter, how it hangs together, and how it is suspended in time?

As everyone knows through accumulated, life-long experience with technology, technology is no stranger to breakdowns. Software is no exception. As users we have learned it is best to acquire basic capacities to cope with its malfunctioning: we for instance take backups, or at the minimum feel appropriate anxiety if we do not take them. Breakdowns are one of the kinds of "infrastructural inversion" events which make visible all that was necessary in the background to keep things going smoothly on the foreground (Bowker 1994; Star and Ruhleder 1996; Star 1999; Graham and Thrift 2007; Denis and Pontille 2015; Denis, Mongili, and Pontille 2016; Denis and Pontille 2019). Software bugs, HTTP 4xx and 5xx errors, crashes and runaway processes, timeouts, security breaches, data leaks, incompatibilities and corrupted files point the attention to what was necessary for software to operate smoothly, most of the time. Repair, fixing, mending and restoration, the activities of breakdowns foreground and make visible all the maintenance that was already there before, busy out of sight in the background. Maintenance is that set of relational material practices which keeps breakdowns from happening, or at least most of the time (well, most of the time). Separation between that which happens after a breakdown, repair, and what was going on before it, what I've called maintenance, is an analytic pairing of concepts, rather than ontological. The ontology is relational: material practice which looks like maintenance from one angle, is repair from another. What matters is that somewhere along the relations, perhaps displaced, maintenance takes place.

How to live in a world which can safely be assumed to be broken? Jackson (Jackson 2014) proposes that a relation of repair and maintenance is the appropriate attitude to take seriously. This attitude of *broken world thinking* does not assume or require that things can be finally fixed, for good. The brokenness is a valuable property of the world itself (Jackson 2014). While much can and indeed needs to be done, no final "fixed" state is reachable – the world will not be ready. Broken world thinking is not a problem-solvers attitude, but that of a maintainer. The fact that the world is broken is not a problem but a predicament. Attending to material, practices and relations, the proposed broken world thinking gives a useful name to a synthesis of lineage of feminist scholarship, including feminist technoscience calling attention to more-than-human vulnerability and fragility, and to think with Anna Tsing and Donna Haraway, how to live and die well on a damaged planet (Tsing 2015; Haraway 2016).

This attitude of broken world thinking permeates contemporary, mainstream science and technology studies as well as anthropology of technology. Research which explicitly calls upon Jackson include Pink et al., who apply it to describe the ongoing material repair practices surrounding broken data (Pink et al. 2018). Kocksch et al. (2018) provide a good, ethnography and vignette based analysis of a care-framing within IT security. Marisa Cohn's ethnographic work among space science engineers at the Cassini mission in NASA Jet Propulsion Laboratory provides a rich, tangible and inspiring background for thinking materiality, practices and relationality of *lifetime issues* in more-than-human entanglements which experience decay of projects, spacecraft, institutions, software and careers over time (Cohn 2014; 2016; 2019; 2021).

Jackson's invitation for *broken world thinking* aligns well with on the one hand with the mathematically proven inconvenient truth about computers which Turing famously derived from Gödel's incompleteness theorem (Turing 1937; Sipser 2012), and on the other hand with the widely shared folk-wisdom among programmers that there are always bugs lurking. And "lurking" is indeed an appropriate term – digital systems are in Derridean terms haunted by apparitions and present absences (Derrida 1994; Fisher 2012; Kjær, Ojala, and Henriksen 2021). Many of the theoretically infinite potential bugs are not actual, but the troubling uncertainty nevertheless is real – bugs are a *virtuality* in the Deleuzian sense (Deleuze 1988), as analysed to the domain of computer software (Mackenzie 2006).

It is worth noting that the concept of maintenance is analytical; the literal words of "maintenance” or “maintaining” are not often the word of choice practitioners use in the everyday, as Rebecca Mossop pointed out (informal communication at *Histories of Maintenance and Repair* workshop at Luxembourg Centre for Contemporary and Digital History, 3th of September 2020), and certainly not in the same meanings scholars cited above have theorised it. Instead the vernacular is filled with other words. In programming these words include, in the English language, “fixing”, “refactoring”, “closing” (opened issues and bug reports) and of course “debugging”. Each has rich discourses and meanings as the *torque* of material, practical relations pull together the biographies of incomplete people and incomplete software artefacts in a broken world. What will we learn if we give room for and attentively listen to the voices of people who care for some of the technologies which make societies endure (Latour 1990)?

3 RESEARCH QUESTIONS

Based on the conceptual framework founded on materiality, practices and relationality developed in the previous chapter, I formulate the following research questions for a qualitative study of software maintenance in a broken world.

When software is seen as a timely rather than an a-temporal one, what potentially might threaten its longevity? Why does it decay if not maintained, and do these reasons differ from needs for maintenance of material technology and infrastructures more generally?

Given what we know about low status, undervaluation, invisibility, and underappreciation status of maintenance work, does software maintenance have similar low status, and if not, how does it tend to differ from maintenance politics generally? Is software maintenance too invisible?

What are main issues in software maintenance, as empirically expressed by practitioners (and to include the feminist reminder: at this time, in this place)? What testimony do they give? What are the maintainers struggling with, what are their concerns and some of their imagined solutions? How do these concerns line up with their personal lives and biographies?

Sociologically, beyond the testimony they give, what do maintainers have got to gain or lose? And at what expense? What is at stake?

What does software depend on, what needs to be in place for it to continue existence? Or to put it more Philosophically, what are the conditions of possibility of the endurance of software?

What might we learn from software maintenance which could be useful for study of maintenance more generally?

Finally, for my own sake, I hope to find a reasonable hermeneutic interpretation of the email I received from Kasper, described at the beginning of this thesis. What kind of an entanglement do I find myself in? What does Kasper's message mean, beyond what is written in its text.

Those are the research questions which guide my venture into this topic. Next, I proceed to survey published research about life alongside technology, and evaluate their insights and research methods.

4 DATA AND METHODOLOGY

4.1 Methods of others. A literature review for methodology

The published literature about living with technology is methodologically often ethnographic in nature. Some of the work focusing on the lifeworlds in which technologies are reproduced and maintained include influential monograph ethnographies such as Traweek’s description high-energy physicists (1992), Orr’s workplace ethnography among photocopier technicians (1996; 2006) and Suchman’s work as an embedded ethnographer at PARC (2012b). The so-called laboratory studies (Latour and Woolgar 1979) also drew on ethnography, although I one evening witnessed dissing of science and technology studies ethnography for superficiality by self-identifying “real ethnographer” (informal personal communication with a respected German ethnography professor and a friend). Henke has expanded the ethnomethodological notion of “repair” beyond discourse (1999; Sims and Henke 2012). Star and Ruhleder described an information system development project from within (1996), further developed into a programmatic invitation for *ethnography of infrastructure* (Star 1999), and further theorised and contextualised with Bowker *Classification and it’s Consequences* (Bowker and Star 1999), a monumental work which has achieved canonical status in both information studies as well as science and technology studies, bridging these two fields.

More recently, Denis and Pontille shadowed service staff by joining repair trips at the Paris metro, focusing on how redesigned signage is practically maintained in the metro system (Denis and Pontille 2015). Repair of “broken data” is described and theorised in (Pink et al. 2018) based on ethnographic and autoethnographic work. We theorised relational ethics in the presence of ghostly apparitions of broken data in Derridean *hauntological* terms in (Kjær, Ojala, and Henriksen 2021). Marisa Cohn's fieldwork at the NASA Jet Propulsion Lab in the Cassini Mission led to fascinating ethnographic insights about living with local legacies, and how biographies and "technographies" torque one another (2014; 2016; 2019; 2021). Cohn's work is particularly relevant for this thesis, as it focuses on software development, production and maintenance under the conditions of received technological, institutional and personal legacies, and their *convivial decay* (2016) over time as the space mission approaches its termination. Part of Cohn's analysis includes accounts of change in software regimes at NASA when unique, bespoke software development and maintenance done in the missions was gradually centralised in commodification processes, a local instance of a well-documented transition in history of software and establishment of "software engineering" as software and hardware were gradually de-coupled into two separate realms (Cohn 2014) through techniques such as programming languages which would run on different individual machines as well as on different machine architectures, often attributed to introduction of the COBOL programming language (Mahoney 2008; Marino 2020; Ritasdatter 2020).

The methods of ethnographic studies are based on long term or midterm immersion in the fieldsites the over months or longer, gathering data through observation and participatory observation methods from the everyday lives of the informants. Some of the research above also reports on study of objects and artefacts, such as diagrams (Cohn, Sim, and Lee 2009; Cohn 2019) and documents (Suchman 2012b).

Other methods of research into life with computer software take the objects and artefacts as a starting point. Media archaeology, inspired by work of German, Kittlerian media theory collects, organises and re-contextualises and re-enacts (e.g. for example exhibits and workshops) historical media objects, including the storage media such as VHS cassettes, tape reels, photographs, MP3 files and game cartridges, but importantly also including the camcorders, amplifiers and mixing desks, cameras, decoding algorithms and game consoles on and for which they were produced and consumed (Parikka 2012; Ernst and Parikka 2012; Piccini 2015)

So-called technography, obviously riffing of the literal name of ethnography, literally "writing people", or "writing culture" (Clifford and Marcus 2010), employes descriptive and interpretive techniques to analyse everyday life of socio-technical and material-semiotic systems (Jansen and Vellema 2011), not unlike the early STS studies and especially the lab studies tradition (Latour and Woolgar 1979). Technographies of software include Taina Bucher's 2012 PhD thesis (Bucher 2012), which has been absolutely formative for myself. App studies implements this approach by reading smartphone user interfaces in a Foucaultian, genealogical way and re-contextualizing smartphone apps in their software ecologies by exposing and visualising inward and outward leading data-flows of apps obscured from the expected end-user (Light, Burgess, and Duguay 2018; Dieter et al. 2018; 2019; Lupton 2020).

Of object-oriented methodologies specifically targeting the source code of computer as its research object is the "critical code studies" program proposed and advanced by Marino (Marino 2012; 2020; Douglass, Marino, and Pressman 2020), and collaboratively performed in the biannual Critical Code Studies workshops (which I partook in 2020). It mobilises the received hermeneutic methods, including close readings of code, in the interpretation of individual computer programs for the purposes of cultural and social critique (Marino 2020). Many of these studies, while apparently selecting source code as their point of departure, come to question sourcecode's position as "the source" (Chun 2008; Couture 2019).

Adrian Mackenzie's hard-to-categorise, engaged methodology includes programming, and aligns with Agre's "critical technical practice" Agre developed in a "tortuous" personal transformation of distancing from the field of AI research (Agre 1997) and draws on autoethnographic, auto-archaeological and didactive-reflexive methods in the process of becoming – through the process of acquiring the expertise, imaginaries and ideologies involved in software praxis of programming (Mackenzie 2006) and machine learning (Mackenzie 2013; 2017).

While all research is practical, some practice-based research methods explicitly draw insight from practical production of technologies. This production side of technologies is at the core of attention in critical making and Critical technical practice (Agre 1997). In artistic research ("Vienna Declaration on Artistic Research" 2020; for a critique see Terpsma and Terpsma 2021; and the neoliberal baggage of artistic research more generally P. J. D. Gielen and de Bruyne 2012) artist-researchers/researcher-artists such as Winnie Soon commit themselves and their artistic and personal self-actualization to involvement in the struggles, constrains, affordances and materialities of imagination and production of computer software for creation of artworks. Knowledge is produced as "a side product" of artistic expression.

Related to the expressive artistic research are perspectives such as critical design and speculative design and design fiction. They elicit, propose and interrogate possible, "what if" worlds through construction of material artefacts such as prototypes, stories (Haraway 2016; Luu, van den Broeck, and Søndergaard 2018), exhibitions, design fiction e.g. product catalogues (Brown et al. 2016), games, pataphysical software (Sicart and Shklovski 2020), interactions and experiences (Ryding 2020) and of course the smart toaster (Rebaudengo 2012a; 2012b). A problem with artistic and design research, both with real or fabulated artefacts, is that while they foreground materiality, relationality and critique, these speculative methods usually cannot help us empirically describe and analyze the practices of living with existing artefacts such as software code.

Multi-sited ethnography investigates social world systems larger than a traditional ethnographic site (Marcus 1995). Ethnographers have also accompanied professionals to their gatherings such as conferences and trade fairs etc. Doing so they take the social dimensions of the professionals seriously (Høyer Leivestad, Nyqvist, and Tunestad 2017), and investigate how local ethnographic sites fit in larger, global, complex spaces (Knauft 2006).

4.2 My method

For my own project qualitative research was epistemologically preferable due to its orientation toward depth rather than breadth in social enquiry (Becker 1996). In light of the methods reviewed above from existing literature, venturing out to do fieldwork seemed like an appropriate choice for conducting my research. This would place me in good continuity with the literature I had studied. I felt ready for participant observation. In my prior scholarly work and in the course of my studies I have studied relevant research literature in STS, done document analysis and interested reading on professional and grey literature in software engineering as well as other documents such as technical standards and deprecation warning messages. I had also conducted discourse analysis on programmer webforums and blogs. This previous work is out of the scope of this thesis, but for the current purposes immersing myself in the language of the fieldsite (Marcus 1995) and it's emic vocabulary (e.g. "technical debt", "bug bounty", "refactoring", "dev/ops") as well as informing myself of some of the contemporary high-profile software maintenance stories such as the Heartbleed security vulnerability which was discovered after years of use in hard to update network infrastructure and the controversy about a maintenance of Node.js package event-stream was helpful preparation for getting ready both participating and observing on the field.

I chose participant observation as my primary method for the fieldwork. The participant observation was complemented by unstructured interviews conducted on-site.

Thus, during 2018 and 2019 I proceeded to conduct participant observation at four maintenance-themed public events. The four events were Maintainers Europe (which I will abbreviate as ME) in Paris, France; Maintainerati (M) in Berlin, Germany, Festival of Maintenance (FoM) in Liverpool, UK; and Maintainers III (MIII) in Washington D.C., USA. These events were selected based on their topic-relevance, and for a balance of focus on maintenance of software and of maintenance more generally. It is worth noticing that all of the events had the word "maintenance" in their title, thereby explicitly and directly pointing to the topic. This is a methodological issue I will return to in discussion. Of the four events two (FoM and the Maintainers series) were familiar to me from academic literature on maintenance and as venues for collecting interests. The other two events were found via these connections.

Funding for conducting the fieldwork was provided by IT University of Copenhagen, as part of a small research project on the topic software maintenance from the perspective of Science and Technology Studies and anthropology.

4.2.1 Participant observation

Participant observation is one of the basic methods of qualitative social sciences and anthropology, and perhaps the most quintessential fieldwork technique (Berg 2007, 4). In participant observation the researchers chooses and enters a fieldsite, and employs their first-hand, subjective, embodied, full-sensory experience as source of research data (Becker 1996; Hammersley and Atkinson 2007). While observing, the qualitative researcher pays attention not only to what is said and done, but to detail regarding the surrounding, atmospheres and interactions between people (Neuman

2007, 287–88), and also includes reflections on their own experience. Notes are taken while on the field, or shortly after (Neuman 2007, 288) for example in the evening when the experiences are still fresh in memory. Fieldnotes vary from jottings to prose, and include observational, inferential, analytical as well as personal content. (Neuman 2007, 288–92; Berg 2007, 197–204). Besides writing, notes might include drawings, photographs, videos or collected items (Neuman 2007, 288–92). Fieldwork and the analysis and write-up period typically alternate, and in the latter the fieldnotes are coded, organised, analysed, theorised and written out at distance from the field (Berg 2007, 207).

Many methods for qualitative analysis of fieldnotes from participant observation exist. The bottom-up, inferential Grounded Theory (Glaser and Strauss 1967; Neuman 2007, 31) is a popular family, with modern adaptations such as Clarke's situational analysis (Clarke, Friese, and Washburn 2005).

Participant observation allowed me to enter the fieldsite, ie. these four events with relative flexibility and to also utilise firstly my preparation as described above, and secondly my prior, personal experiences of programming while watching and listening to the participants at the four events. While discussions in the events were typically on a relatively general level, they would occasionally spiral into language saturated with technical terminology. Having some familiarity and first-hand experience with the praxis of programming allowed me to follow the intended meaning most of the time. I did not *a priori* worry too much about the anthropologists risk of "going native" (Engelke 2000, 850) and becoming blind to my own position as a researcher who ultimately is alien to the field, and as an outsider can and must eventually exit, leaving the "native" informants behind (Hammersley and Atkinson 2007, 94–96). Besides my fieldwork being short in duration and it's immersiveness limited, the necessary analytical distance and *estrangement* would be regained in engagement with theoretical STS literature, and in reflexion of the haphazard nature of my own programming and software maintenance experience, and that my own life does not hinge on it as a profession, vocation or as identity.

4.2.2 Unstructured interview

Unstructured aka. unstandardized interview (Berg 2007, 93) techniques are open, thematic discussions with informants. In contrast with more formalised interviews or surveys, unstructured interviews are particularly suitable for research where the right questions are not known ahead of time (Berg 2007, 94). Rather than extracting answers to predefined questions, the interviewee and the situation are allowed to participate and even lead the course of the interview, and the topic is allowed to shift. As the interviews are not based on predefined questions, requirements of shared language and common interpretation of the questions are relaxed.

Given that only a small amount of interviews were expected to be done during fieldwork, unstructured interviews were considered to be preferable to more formalised, structured interviews with pre-formulated questions and a schema for a number of reasons. Firstly, no statistical power could anyway be gained from the small sample, as its size was expected to be small at each of the events. Secondly the interview situation during the four events was not under the control of the research. Thirdly the research aimed to investigate the emic vocabulary of the informants. Unstructured interview would allow the field to express its own language, rather than require the interviewees to adapt to a language and schema the researcher would have constructed, imported and imposed.

4.2.3 The method assemblage

Navigating a level of participation or selecting an interview style is a balancing act. So are all decisions about research methods. While these decisions appear to be owned by the researcher, and consequential mostly for their own epistemological work, there is more to be considered. In his book *After Method*, John Law (2004b) argues that social science research always involves dealing with *mess*. Rather than being outside of this mess looking in, the researcher is always inside of it, co-constructing the mess with choice of method and their composition in *method assemblages*, the always unique and situated collection of practical research actions. Being constitutive of the worlds they are applied within (Law, Ruppert, and Savage 2011), the choice of method cannot be seen only as a practical, epistemological question for answering research questions, but always-already political.

Law's argument, influential in Science and Technology Studies, aligns well with and is informed by feminist scholars who have long argued against objectivism, and the violences and omissions that the desire for so-called God-eye view, a falsely impartial "view from nowhere" necessarily perform. Feminist objectivity builds on partial views. (Haraway 1988). These feminist correctives to study of science and technology have contributed to co-called "post-ANT" developments of Actor-Network Theory and other STS devices (Denis and Pontille 2019), of whose invention and amendment Law has contributed to.

In my selection of two relatively open methods – participant observation complemented by unstructured interviews – into the method assemblage I tried to remain sensitive to multiplicities, indefiniteness and flux (Law 2004a, 14; 2004b, chap. 6) during both the fieldwork and the subsequent analytical writeup phase. I tried firstly to be a good anthropologist by setting myself up to be surprised (Winthereik 2019). I found that preparation work both precluded surprise by giving me expectations about the topics, concerns I would encounter, the people I would meet and the interactions I would have. At the same time preparations were the enabling condition for surprise, heightening and accentuating the moments when my expectations proved false or incomplete. Secondly I aimed for a thicker description (Geertz 1973) than what methods of surveys, interviews, archival research or object analysis would have afforded for.

While a research design surely informs how research is conducted, in practice research is always performative. Research is done. Research is never fully determined by the pre-conceived plans, but is always *situated action* (Suchman 2012b). Research is an ongoing unfolding of a performance. I will next characterise the four attended events at more depth, and then proceed to give an account how my research design unfolded during the events.

4.3 Description of the four fieldwork events

In the following I will briefly characterise and contextualise each of the attended four events. I will then proceed to generalise them and place them in two categories, one focused on software maintenance and the other on maintenance more generally. This will be useful for framing and contrasting the software specific questions within published literature on maintenance, repair and care.

The first event, Maintainers Europe was a minor track at API Days, a large software development industry event which is organised globally multiple times per year. The track focused on maintenance questions was arranged in a separate room upstairs of the fair, organised in connection with Maintainers, a network of scholars and activists focusing on various aspects of maintenance and repair. The second event Maintainerati was organised by a small group of people affiliated with software development as a side-event on the day following the dominant software development platform's GitHub annual event in Berlin, at the same venue. The third event, Festival of Maintenance was, as the name suggests, a celebratory event in the UK, and on its second year in 2019 it was organised in a post-industrial venue in a downtown district of Liverpool. Finally Maintainers III was the third instalment in a conference serie organised by the academia-led Maintainers network mentioned above, and took place over four days at a University campus.

4.4 How I did participant observation and interviews aka performing the method (assemblage). A post-facto account.

At all of the four events I did my best to try to establish and stabilise a style of pen and paper based, written and drawn fieldnote taking at all of the four events participated. In my notes I paid attention to what was said, references, and which items recurred in multiple discussions. I also tried to pay attention to who was in the room as it appeared to me at the time and to their occupation as they referred to it, while trying hard not to make too many assumptions about people's cultural, ethic, regional or class background or to their age (without a doubt I failed to be ignorant of this). My notes also include personal reflections about my state of excitement, inspiration, bio-needs and occasional awkwardness. At Maintainerati I additionally joined the invited ranks of collective note-taking, and passed them in a slightly abridged form on to the organizers which were shared in a repository which is available online upon registration and collected and synthesized in two publications, as session notes ("Maintainerati Berlin 2019 Session Notes" 2020) and a report (Lynch, Taylor, and Goodman-Wilson 2020). I took some active part in the discussions both in the sessions and more personally in-between, as well as in the various coffee and snack chats, lunch breakaway groups, afternoon ice-creams, dinners and evening bars with drinks and food, as well as backchannel chatter on Twitter.

I chose to take my notes on pen and paper, and transcribed them onto the computer later the same day or week. I can better pay attention when writing on paper than on a computer. The straightforward, familiar and powerful tools of pen and paper also allow sketching, and in my notes I included drawings and small maps of how people positioned themselves in relation to others, and I also sketched impressions of some of the scheduled speakers. I found the latter to be a powerful memory aid later on, reviewing my notes. Finally pen and paper also free me of contemporary concerns of battery life and positioning myself more freely without those needs in mind. Pen and paper was a fortunate choice from the perspective of blending in, since most of the participants at the software maintenance events, perhaps surprisingly, did not use a computer during. When they took notes, they did so almost entirely using pen and paper. I transcribed my notes onto computer files either the same evening after the event, or on the following day. I found transcription of an entire day worth of hand-written notes to be rather exhausting, as suggested by Berg (Berg 2007).

The next chapter provides an analysis of the fieldwork data and describes the main findings.

5 ANALYSIS AND FINDINGS

To analyse my fieldwork data described in the previous chapter, I begin this iterative chapter by characterising each of the four maintenance events, classify them, and through contrast and abstraction show how they were similar to and different from one another. I will then present an ethnographic analysis with two vignettes which synthesise, thicken and concretize the fieldwork, and serve as material for the final part moving toward theoratization. Finally I move to a discourse analysis which contextualises the maintenance events in contemporary life alongside software.

5.1 Categorical analysis

As a first step, I begin the data analysis with a classification. This will be crude but productive towards more nuance below. Classical studies of meetings classify them based on whether they are scheduled beforehand, or unscheduled (Schwartzman 1989, 61–64). All the events of my fieldwork were scheduled with the date and place announced, thus this ontology is not useful for producing a classification. An alternative, emic classification arising from my fieldwork is separation between two classes of people, namely “the practitioners” and “the researchers”. Following these emic terms themselves would have virtue, as lateral concepts (Gad and Jensen 2016). I however reject this opportunity for now because of the multiple slippages between these received, identity-based categories, plus my own positionality as both a software maintainer and as a researcher of software maintenance (see chapter 1). Instead, I use a binomial classification based on whether the topic of the event was maintenance generally, or maintenance of software specifically. Here I must beg for the patience of the reader to follow along this rather simpleminded and tedious rehearsal of classification as a starting point. While simple, it will highlight key differences early on, and already hint at some specifities of materiality, practices and relations around software, which are the proper object of my study and which will be addressed below.

5.1.1 Event programming

	Generic maintenance (FoM, MIII)	**Software maintenance (ME, M)**
Event program	Curated program	Proposed and voted on the spot
Event content	Prepared presentations	Focus group discussions

Table 1. Organisation style across event types.

Of the four events included in the fieldwork, the two generic maintenance themed events (Festival of Maintenance, Maintainers III) had a curated programme of prepared presentations, announced with presenter names and presentation titles to the potential audience ahead of the event (see Table 1, left). FoM was a single-track event with a wide variety of presentations ranging from museology to social science research, from service design to bridge engineering and tailoring. At Maintainers III the bulk of the programme aside from introductions, plenaries, keynotes and summaries was divided to parallel tracks on the subthemes of information, transportation and software maintenance. Both events concluded in a panel discussion with invited panellists. Such an organised and communicated format is not uncommon in academic settings.

In contrast, the two software maintenance oriented events (Maintainers Europe, Maintainerati) were organised in a so-called “unconference” format with no pre-organized programme (Table 1, right). “Unconference” is an entire family of event types and as the name suggests is characterised negatively against a usual conference. At both unconferences besides the welcomes, introductions and some keynote presentations done in plenum, the participants were tasked to propose topics in the morning, vote on them, and then proceed to discuss them in round-table sessions lasting approximately one hour each. Lunch and other breaks structured the day at both of the events. An open wrap-up session concluded both of the events. Such open, relatively self-determined formats are not uncommon in software development and “tech”.

While there were differences in regards to how the event content was planned and performed, there were also similarities. On one hand the programme consisted of prepared presentations delivered from a stage to an audience, and on the other hand a were improvised in participatory topic-proposal, collective selection, and then individual choice of which sessions to attend. The obvious similarity between all four was the relatively insular separation of all of the events with regard to time and space; there was little question when the event started and ended, and where the boundaries of its site were. For any of the events to run smoothly, an infrastructure of a commody event venue, catering service and service staff was a prerequisite.

5.1.2 High-level content analysis

Next to characterise and further contrast the two event categories, the following table (Table 2) sketches out a high-level content analysis of fieldwork observations, focusing on their differences across the dimensions of the role of sharing exemplars and experiences, the scope of concerns and timescales addressed, their problem orientedness, and who the participants considered to have agency over the relevant concerns in maintenance.

	Generic Maintenance (FoM, MIII)	**Software maintenance (ME, M)**
Shared exemplars	Few	A few recent ones
Shared experience	Typically no	Assumed yes
Scope of concerns	From “my work” to capitalism	Political economy in “tech”
Timescale	Years to decades	Months to years
Problem orientation	Problems, generic solutions	Problems, specific solutions
Agents of change	Public sector, moral subjects	Products, infrastructure, moral subjects

Table 2. Content dimensions across maintenance event types.

Both the presentations at the generic maintenance events (FoM, MIII) and the discussion at the software maintenance events (ME, M) put forth examples and exemplars of when maintenance had failed, succeeded, or become contested. At both, they served as narrative devices and crystallizations of the issues being discussed. At the generic maintenance events the exemplars were described more thoroughly, assuming that the audience was not already be familiar with case or story. This reflected the presenter/audience roles of the event structure, and specific domain expertise of the speakers which was to be explained to the audience unfamiliar with the domain. In contrast at software maintenance events many occasions when examples were put to use, a much higher level of familiarity was assumed, and sometimes only a minimal index toward an recent story was provided (e.g. “with Heartbleed”, “the new GitHub feature”); the participants were expected to either be familiar with the specifics of the case or possess sufficient background knowledge to infer how the story unfolded. This practice asserts ingroupness.

Relatedly at the software maintenance events a shared life-experience too was assumed between the participants. These were events for “us”. No such expectation was put forth at the generic maintenance events. While occasionally ingroup-checks were performed at the software maintenance events (e.g. “you know the reverse Conway’s Law, right?”, or “as product managers always think, you know?”), they were typically skipped and the other participants were assumed to be able to extrapolate from an unspecified programmer experience to make sense of what was said about maintenance; to synthesise the mood, “everyone knows what a refactoring sprint feels like” and “we’ve all seen a burnout up-close”.

Both the shared exemplars and the shared programmer experience at the software maintenance events point to an insightful reversal regarding the relationship between culture and stories (Orr 2006); rather than the matters of concern in software maintenance becoming understandable through stories, the stories themselves become understandable only because the programming matters of concern are already familiar for the participants. In this mode of sharing, rather than new knowledge being communicated to an ignorant audience, existing understanding is called upon and put to use.

With regards to the scope of concern and timescale considered relevant for maintenance, the two event types differed considerably. The generic maintenance events, unsurprisingly, covered a temporally wide range. On the micro side, the informants shared personal observations about life with artefacts, and touching intimate accounts of mending and care. On the macro side they covered academic analyses of global late-stage advanced capitalism and the climate crisis in its wake, the existential anti-colonial and anti-racist struggles under the contemporary world order. Of course the Anthropocene loomed in the background ominously (see Hallé and Milon 2020 for discussion of the name).

This was clearly contrasted with much more constrained scope of concerns and interests in discussions around software. While personal experiences were shared in this context too, the discussions rarely reached beyond the social, technological or economic world which didn’t directly affect the participants and the material of their maintenance, ie. computer software. The microscale was also absent, and I heard no detailed descriptions of how some lines of code were modified. A hesitant self-constraining of the scope of discussion to software was evident; it remains unresolved whether this ought to be best interpreted as modesty, or as short-sightedness and failure of social and ethical imagination.

The above observations regarding the scope of concerns is anything but surprising, but rather an artefact of the method assemblage of my analysis – if one’s analysis calls one of the event types “generic”, it will come as no surprise that contemporary generic concerns of Western intelligentsia (e.g. rampant capitalism, post-colonialism, climate crisis) will surface.

Moving on, what is more interesting are the differences in problem orientation and attribution of agency, ie. who the agents of change are considered to be (Table 2).

All the events shared problem-orientedness and focused on the material failures, discursive omissions and political struggles of and about maintenance. In software maintenance events participants were quick to reduce the problems to something to which preliminary solutions could be formulad, even if for the sake of debate. Typically problems were framed as failures of incentives, and then pros and cons of alternative re-designs of incentive structures discussed. Examples of such reduction of problems to engageable, possible solutions included bug bounties, how a programmer's contributions to Free and Open Source Software form a visible part of their public profile, how the experience of stress can be managed by reducing the flood of notifications a software maintainer gets, redirecting flows of money and prestige to more infrastructural software projects on which the visible ones depend on, how documentation work, project management and community involvement might be made visible as valuable contribution in addition to programming of code. According to my participant observation at these events the agents of change who would be able and expected to respond to the problems were imagined to be relatively close and familiar, such as individual programmers, the companies and software projects they worked for, their tools, and the platforms through which programming work is conducted. Relatively few appeals to public institutions were made; they were configured to be outside of the scope of the relevant technological imaginary (Balsamo 2011; Suchman 2012a). This configuration of the technological imaginary was in stark contrast with conclusions at the generic maintenance events, where large-scale actors such as national governments, legislative bodies and megacorporations were seen as the most relevant agents of change to improve conditions of maintenance. There, the matters of concern were framed through failure of policy and democracy, and calling upon and re-centering the democratic institutions as agents of change was the most relevant course of action.

However at all of the four events an appeal to one very specific agent of change was repeatedly made (Table 2). This called-upon, interpellated agent was the morally virtuous Subject, who would by necessity of their virtuousness recognize that maintenance was something to appreciate, cherish and improve; something good. I call this *the moral work of maintenance*. The concept is ambivalent, and both descriptive and prescriptive. Maintenance is morally good and praiseworthy, and as care maintenance is something one *ought* to do – literature on ethics of care (Tronto 1993) is explicit about it. Failure to appreciate material maintenance and repair, and more morally loaded *care*, ie. neglectfulness, is morally corrupt and blameworthy. "What kind of person would not value repair, maintenance and care?", one might ask. This moral work ties back to the awareness-raising and self-recognition purpose of these events, and valorization efforts of the work of those involved in everyday repair and maintenance practices.

Who then has the ability to live up to these morals, and what kind of material is able to be maintained? In other words, what does it take to be able to live up to the strongs demand to maintain material things, and which material things present themselves as able recipients maintenance? What are the links between morals and ability, and moral and agency? In what circumstances can relations of caring for the material world practically be exercised? The plot is thickening…

5.1.3 Representation versus reflexion

Building first on the high-level content analysis above (Table 2), a number of similarities and differences can be observed with regards to whether maintenance was seen primarily as an object of representation, or an object of reflexion.

The generic events were representational in the sense that the speakers aimed to present maintenance to the audience from with-in maintenance praxis, part more analytical from a position from with-out (Table 2, left). This was performed by lower assumptions that the audience or other discussants would hold first-hand experience of material maintenance, and be able to understand examples and stories against that background. Instead, the examples were explicated and told about through spoken narrative supported by photographs on presentation slides, and the audience would take as given what the expert said the matters of concerns were. Maintenance was thus made clearly imaginable for the audience, who would nevertheless remain outside of maintenance of garments, museum collections, data platforms, road bridges, university campuses or hydro-electric plants. Per the structure of the events, each presentation gave a clearly communicated image, and the presentations *in toto* invited the commonalities and differences of maintenance and to apply *broken world thinking* (Jackson 2014).

Rather than providing a *smörgåsbord* of representations, the software maintenance events oriented towards the individual and shared experience of the participants from within the community of practice (Wenger 1998) and drew heavily on first-hand life-experience and tacit knowledge of the in-group (Table 2, right). As such, they set up what Agre has called critical reflective practice (1997).

Regarding knowledge-building, the two event types differed in sharing information beyond the events themselves. Of the two generic maintenance events, the entire formal programme of Festival of Maintenance was streamed and archived on the public Internet, and at the Maintainers III many of the presentations were either published or work-in-progress publications, academic or professional in nature. At both of the two software maintenance events the Chatham House Rule was announced (ie. information at the event could be used freely, but information not traced to the participants), effectively creating a safer and less public space. Views and material politics of this knowledge work and accumulating cultural knowledge in artefacts will be a key element of my analysis below.

All the events across both types were marked by a basic recognition of the value of maintenance, and community-building was a key meta-purpose at all four of them. The moral work of maintenance was practised via appeals to the absence of an crucial topic from discourse and about which the participants had something valuable to say, the urgency of the topic, various stories of lacking, failed and heroic maintenance, and social construction of archetypical maintainer figures which would serve to represent the people and work usually made invisible.

5.1.4 Summary

While reductionist and even naïve, the categorical analysis above in this subsection has characterised the topic of repair and maintenance, what is at stake, and shown what the overall ethos, pathos and logos of landscape which frames concerns of software maintenance. The grouping to generic and software maintenance (see Tables 1 and 2) is not without complications. A first complication for the classification comes from that at Maintainers III one of the tracks was focused on software maintenance, and was organised as a mix of traditional discussion-based “unconference” and traditional presentation-based “conference” formats. A second complication is that the events are anything but independent of one another, but rather a network of people interested in these topics would tend to organise as well as attend the events. I myself am invested in this, and the researcher, moving from situation to situation, project to project, is themselves a bridge between them (Ahmed 2019).

With those complications to the categorical analysis, it is time to jettison the classification and start moving toward further substantiation by giving thicker descriptions of the happenings in my fieldwork.

5.2 Ethnographic analysis

To describe what happens in these events with more depth, I have constructed two *ethnographic vignettes*. Vignettes are an analytical method in qualitative research, and particularly important in ethnography (Hammersley and Atkinson 2007, 196). Vignettes are a literary tool with which the ethnographer transports the reader to the fieldsite, and invites the reader into the subjective, first-hand experience gained there. It is important to understand that vignettes are constructions – they do not aim to be objective or exhaustive recountings, and are often synthesised from multiple events and observations. The literary genre of narrative is often chosen. After a vignette, the ethnographic account provides an analysis and situates the vignette in the wider context of the fieldwork and in theory.

I present two vignettes, titled *Not everyone is Kubernetes* and *Maybe one more day per week*. Each is followed by a brief analysis where I make explicit how they draw from the categorization above.

5.2.1 Vignette 1. But not everyone is Kubernetes!

> The relieving release of energy is palpable as the circle of seats breaks and people start standing up and collecting their bags, water-bottles and jackets. The clutter of chairs mixes with voices of the two dozen participants – an intense one hour discussion about community engagement has just finished. The session was productive and hearing about the impressive efforts of a long-standing, well known and respected open source software company uses to onboard new maintainers was inspiring for everyone. Not only do they translate beginner documentation to a number of languages such as Spanish and Portuguese and employ mentors to help everyone feel welcome to the codebase and have a good developer experience, but the charismatic, colourful thirtysomething

> community manager said they also aspire to mention every single contributor in the credits by name! Impressive.
>
> The session went a little overtime and eyes are veering around the conference hall looking for orientation, snacks, toilets, friends and a bit of a stretch. As I squeeze through the doorway with Uwe, a middle-aged man with a well-worn casual shirt and a dark shoulder bag, he sighs to me with frustration in his voice “but not everyone is Kubernetes!”. Before I get the chance to ask him what he means, he makes eye-contact and goes on to explain that he is the sole remaining maintainer of a once popular software written in Java. There are no teams of community managers; there is no longer even a team; it’s only him now. We walk abreast towards the hall where coffee is already being served, and I fail to find words of comfort.

Uwe’s frustration highlights the dissonance between the æsthetic and ethical aspirations, and the lived realities in software maintenance. Software is not one single thing (though could be theorised as such – topic of my future research), but typically understood as individual “products” or “projects”. These products and projects are of huge diversity. While household names such iOS, Windows, Chrome, Firefox, Fortnite are familiar in popular culture, “googling” and “photoshopping” verbs in the vernacular, and software applications like Tetris and Super Mario Bros connect across generations. In the world of programming some bodies of code are widely known, such as the container system Kubernetes Uwe referred to. Most software projects are, however, very small; software has a very *long tail*. The WordPress plugin I told I maintain in the introduction of this thesis is somewhere far down this long tail of small projects, as is Uwe’s Java application.

The session we left with Uwe had toward its end turned to a community-manager from one of major corporation-backed open-source projects describing how they onboard volunteering contributors, and how important good “developer experience” (DX) is especially for newcomers. Although interesting and valuable, these are not relevant concerns in the long tail for the many software maintainers who are not working at such a large scale. The frustration and helplessness Uwe expressed was that of alienation – he had joined a meeting of software maintainers, but the discussion had alienated him further. He could not live up to the ideal moral work of maintenance. The software he maintained was written in a programming language no longer fashionable and with a reputation of being unwieldy, cumbersome and “not fun”. While there had earlier been others to develop and maintain it together with him, they had now left and he found himself to be the one who has to stay behind and continue a life alongside this body of software.

This vignette foregrounds the entangled biographies of people and biographies of artefacts, and how they are pulled toward one another through the force of *torque* (Bowker and Star 1999; Cohn 2016; see Johnson 2016 for a first-hand account of being left behind to maintain a crucial piece of infrastructural software). The scales of concern are practical and personal – while Uwe might be interested in valuation (or lack of) maintenance, he has joined the event not to learn about principles, policies and “nice to have”, secondary issues of diversity and inclusion, but to find peers who share the first-hand experience of living alongside a small marginal software project, specific solutions to practical problems at hand, and sharing the workload of responding to user requests, fixing bug backlog and keeping the existing material artefact in decent working condition.

5.2.2 Vignette 2. Maybe one more day per week

> It is close to midnight, and I'm leaving the party. The warm, dark urban late Spring night engulfs me. I am satiated with culinarily quite interesting artisanal cocktail snacks sponsored by the hosting megacorp whose annual event for software developers was held earlier during the day. For this I am grateful, as I saved 3€ on dinner döner. I am leaving the party together with Stefan, a sympathetic ponytailed and bearded man in his mid thirties. As we step out, we pause on the street to finish our conversation before getting a decent sleep for tomorrow's maintainer meeting. He shares with me something he has hesitated to tell his boss: that he would like to reduce his work from four days to three, and attend more to his pet project, a game world creation tool.
>
> It had all started as something just for himself in his computer game creation hobby. The existing tools for world-building were too expensive. And like many programmers, he had shared the code online. During the past decade many other game creators had started using it. Growth of the indie game culture was the reason for the uptake, he tells me. While now there might be enough crowd-funding from small game development studios and hobbyists that he could afford a second day a week on it, the boss at the "real job" had already been a bit unhappy to lose his expertise one day a week. Expressing the wish for moving to a three-day workweek might cost him his enjoyable programming dayjob. But the wish to better maintain and develop this world-building tool on which many computer games and their creators made themselves dependent is strong and felt like the right things to do.
>
> That hesitation wouldn't be resolved tonight he concluded, as we wished good-nights and bis-danns, and walked our own ways.

In this vignette we observe many of the characteristics of the software maintenance events as analysed above in the categorical analysis in the previous section. Software maintenance might mean being involved with multiple software in parallel or after one another. "Side-projects" and work mix and morph into one another over time – a side-project might become a job, or as in the case of Stefan a job might become more of a side-project. At the events this articulation work to deal with the flexibility of programming work came visible in two specific ways. On one hand the socio-economic structures of recognition, resourcing and compensation of maintaining existing software were a topic that surfaced repeatedly, and many of the topics suggested in the morning addressed the economics of channelling money to would be needed and deserved. On the other hand the articulation work was addressed in debates about burnout. While both aspects, money and burnout, manifest themselves individually such as Stefan negotiating with his boss, the concerns are also collective and structural issues. These are signals of precarious and entrepreneurial political economy. At the software maintenance events a number of new products to channel funds to maintenance were discussed, and the companies also had their staff present. These included startups as well as the corporation running the dominant software development platform. Before these products existed, Stefan had been able over time cobble together a number of revenue streams to fund the first, and now potentially the second day of week to dedicate to his side-project. While Stefan was negotiating his partial success, Uwe in the first vignette had fallen into margins of legacy with his project and was not in a position to reflect on his subjecthood, agency and hesitations between available choices. Stefan, one could dramaticize, was tormented by the predicament of finding himself a victim of his own success.

Here, the relationship of maintenance goes both ways. As Stefen is debugging, refactoring, simplifying and extending the tool, the tool is providing him with a channel of reflection and self-expression, a social position as owner and maintainer, pleasure and a sense of purpose, as well as sustenance. We did not discuss personal economics with Uwe, but according to my interpretation the software was struggling to support and maintain him.

Stefan himself reflected on his hesitation – as his confidence and crowdfunding had grown, perhaps he was ready to end it. Risky negotiations are part and parcel of anyone’s life, but particularly characteristic of the labour relationships of those who do maintenance work (Vinsel and Russell 2020). While hesitating at the risk of losing his pleasant and rewarding job in pursuit of making the material maintenance of his “pet project” a larger part of his working life, Stefan’s affect does not reflect lack of agency. Exactly the opposite is the case: his affect stems exactly from the fact that he *can* exercise agency. While anything but straightforward, the participants' sense of emancipation was in stark contrast with the analysis of maintenance as low-status work without much possibility of reconfiguring the relations in which the maintenance practices take shape.

We see in these vignettes that software maintenance is category-fluid, and articulation and re-production of categories of “work”, “hobby”, “development” and “maintenance” themselves is part of software maintenance. At the events the participants temporarily froze “software maintenance” as a category stable enough for the sake of having a theme for the meetings and concerns. A negotiated and useful “as if”. These practical and situated stabilizations are part of ethnomethods (Garfinkel 1967; 1996) of programming. While a programmer might have the identity or job title of “developer”, much of this would consist of maintenance of existing software. When the “software maintenance” meeting ends, and everyone returns home to attend to their material artefacts, “maintenance” melts back into the individual, local and situated practices of life alongside computer software.

5.3 Discourse analysis of software maintenance

I now turn the attention of analysis to the discourse at the two software maintenance events, foregrounding what the participants discussed while further backgrounding the organisation and formal schema of the events this chapter started with. As described above, these so-called unconferences had no pre-announced tracks. Rather, the participants were invited to put forth topics in the morning, after introductions and keynote talks. This was followed by a consensus forming process, a mix of voting and negotiation about merging. After the programme had been set, the participants would then self-select which of the sessions they would partake in, and to what extent. Researchwise this procedure warrants paying some attention to what was proposed, as it sheds light on what the participants hoped to get out of the event before it happened, and what they imagined would be interesting to whoever was present. It is worth noting that the sample size of two events totalling a few dozen self-selected people is of course not representative of pertinent – or even relevant – topics of software maintenance. Representativeness is not a stifling concern in epistemology of qualitative research (Becker 1996). But I claim the sample of proposed themes most certainly represents the imaginaries and the issues of software maintenance as those present at this time and at this place narrate and configure it (Suchman 2012a). Contrasting the proposed themes with the discourse in the two generic maintenance events on one hand, and published research literature on the other ought to lead to insights.

At smaller Maintainers Europe a handful of topics were proposed in the morning. At the larger Maintainerati, no less than 40 topics were put forth, merged to 33 eventual sessions. The topics formulated onto colourful post-it notes often incorporated emic technical language, such as “real-time communication in OSS (Open Source Software) communities” (*real-time* in contrast to delayed signals), “making ourself obsolete” (*obsolete* being a pejorative term for old, outdated and unnecessary systems), “issues with issues” (an *issue* being a reported bug or feature request), and “getting almost 100 people started with their first OSS contribution” (*contribution* being an item of productive work considered valuable, and to which I return below) to name four examples. In synthesis, the topics cluster to cover economics, communication, as well as socio-cultural issues.

5.3.1 Two themes differing between generic and software maintenance: recognition of labour and distribution of labour

Two themes present at the generic maintenance events as well as published research literature, but framed otherwise in software are firstly the questions concerning recognition and valuation of maintenance work, and secondly their distribution.

According to published literature on maintenance, it is typically poorly recognized and undervalued as work (Graham and Thrift 2007; Denis and Pontille 2015; Mattern 2018; Denis and Pontille 2019; Vinsel and Russell 2020; Lindén and Lydahl 2021). The roots of this misrecognition are by now familiar – the gendered, classed and racialized structures privilege the activities of men, activities of upper and middle classes, and activities of white people. Additional, related hierarchies favour the novel and “innovative” over the existing and familiar technologies (Vinsel and Russell 2020) and over infrastructure (Star 1999). Beyond social concerns, the human need for recognition as a philosophical discussion traces back at least to work of the 18th century German Philosopher Hegel, but underlies current discussions of “identity-politics”. (Vinsel and Russell 2020). Some professions are valued over others, for instance doctors are held in high esteem at the expense of the necessary and hard work of nurses, struggling for recognition and visibility in standardised reporting and management mechanisms (Bowker and Star 1999). Similarly, engineers are appreciated over technicians. Vinsel and Russell sketch a Foucaultian genealogy, ie. the ideological history of the low status of maintenance illustrates how status hierarchies of various jobs is a social construct produced and reproduced throughout childhood, school, higher education and beyond (Vinsel and Russell 2020, chap. 6). In an attention economy, activities open for economic speculation trump over the stable, familiar, everyday activities. Low-status jobs are poorly paid (if paid at all), and left out of grand visions, future imaginaries and success stories told afterwards. Much of this research has drawn on foundational work of feminist scholars about vocational hierarchies and invisibility of domestic work and the roles of technology at home (e.g. Cowan 1985).

The distribution of maintenance follows the same pattern. It tends to fall on the shoulders of the subaltern and further exacerbate their plight and indeed their invisibility (Vinsel and Russell 2020). Maintenance and maintainers are stigmatised in circular logic of a self-fulfilling prophecy; maintenance is dirty work, and by implication those involved in it must be dirty too.

This underrecognized status, even stigma of maintenance as activity and those employed to do this necessary activity was a key premise for all the four events of my fieldwork. While the software maintenance events too aimed to raise awareness and build community of practice, the low status of maintenance however came to be contested. According to the informants, unlike in the published literature, software maintainers do not necessarily occupy the lowest rungs of status hierarchy. While often menial and typically invisible, software maintenance sometimes comes to be seen as high-status work, sometimes even top status. This is an attribute particularly in open source software where ownership of the material code and the project around it is considered to emerge from Lockean theory of property ownership from labour (Locke 1689). The (questionable) principle of meritocracy supports this process of ownership, as does experienced practitioner’s acceptance that building software is hard and laborious. Maintainers are the owners, and the owners are the maintainers. That’s the ideology. Things are not so straightforward, as my fieldwork reveals.

Gaining ownership-through-maintenance involves technical mastery, long-term commitment, and good oversight over the entire body of code and it’s local socio-material microhistory. While at a meeting of software maintainers this happens to be self-congratulatory, the elevated, centralised subject-position of maintainer was anything but unproblematic – many stories of crude, abusive, elusive or simply gone (including burned out) maintainers were shared and multiple sessions were proposed about the topic.

Other stories which complicate the maintainer-as-owner picture emerged from life-stories of maintainers who find their own biography and the material software artefact they maintain pulled being together by a torque they cannot escape (Bowker and Star 1999; Cohn 2016). This theme appears in the first vignette in the previous section with Uwe. The point of being bound to an artefact is powerfully made in a blogpost titled *All Software is Legacy* by Lee Johnson, the burdened volunteer maintainer of the influential Perl module *CGI.pm* which has served as internet infrastructure since 1990s (Johnson 2016). The observation that maintenance can be lonely and hard to get out of is consistent with the underappreciation and invisibility theses, but not necessarily congruent with the low-status thesis. “It can be lonely at the top” as one of my informants put it.

Unlike in many other relationships with material artefacts, building and maintenance are not clearly separated in programming. We have architects and facility management for buildings, private development projects and public maintenance of road infrastructure (Vinsel and Russell 2020), typeface designers and repairman for signage (Denis and Pontille 2015), operations theatre and aftercare in hospitals (Mol 2003; Mol, Moser, and Pols 2010), farmers and vets at farms (Law 2010), factories and and garages for cars (Dant 2010), designers and repair shops for smartphones (Nova and Bloch 2020). While in software too we find a difference in development and operations, in other words programming and administration of installations done in quite separate circumstances, maintenance of the the source-code-as-source often falls on those who first wrote it (for critique see Chun 2008 on “sourcery”; c.f. Couture 2019). In software, maintenance is displaced from operations, with solutions to local problems often requiring a detour through the “upstream” maintainers; while a bug can possibly be fixed at the installation, it is preferable to solve it in the software source code of which the installation is an instance of.

The alternative framings of software maintainers firstly as skillful, dedicated caretakers or even romantic “gardeners” (emic term), and secondly as tragic figures bound to their artefacts (e.g. Cohn 2016; 2019) challenge the widely documented notion of maintenance as drudgery for low-skilled, expendable individuals doing necessary but unappealing work. Maintainers instead become an obligatory passage point (Callon 1984), and maintenance as a nexus of material, practical and relational power. Following the actors through these networks and telling biographies of people and software yield different kinds of stories than definitions focused on assumed stable subjects of objects. These are stories of becoming rather than being.

A recurring theme in the discussion at the software maintenance events was about the dynamics of maintenance positions, or more precisely funnelling people from “drive-by contribution” (emic term) to the high-status, ownership based long-term maintenance as the ideology requires. No less than a third of the proposed session topics addressed these dynamics explicitly. The implicit premise is the need for maintenance people. Onboarding maintainers was seen to provide a break for the lead maintainers, career opportunities for juniors, and a source of diversity, including gender, class and racial/ethic diversity to software projects including their apparent primary object of maintenance, the code. Thus, maintenance became a *Ding-an-sich*, and itself an object to be cared for.

5.3.2 Two themes exclusive to software maintenance: flow of information and tooling

Two themes visible in the data of my fieldwork but poorly covered by published maintenance literature are information flows and tooling. Both themes appeared in the topic suggestions the participants put forward at the starts of the events, as well as in my content analysis. If computer systems are seen as tools for managing information flows, this should not be a surprise. Besides "information" in the abstract, it is worth considering the specific content and flows of information, as discussed by the participants. Paula Bialski has analysed ethnographically how code review process organises communication of programmers, machines and their organisation in the context of a large software development company (Bialski 2019). Above, I already analysed the flows of people into and out of maintenance to be one of the central topics, and how its dynamics differ considerably from maintenance studies generally because of the prestigious and powerful status maintenance within the socio-cultural realm of software.

From the perspective of the material software project, to onboard, retain and retire maintainers is to manage information flows. Onboarded new "contributors" bring needed "hands on deck" to deal with the mounting maintenance burden by reporting bugs, testing, writing documentation, responding to users questions and requests, and reviewing code and in the case of open source software, engaging with the community of users, power-users, and developers. People therefore are made to be information. As knowers and skilled experts, maintainers literally speaking incorporate the experience of putting local know-how to practice. Retaining this hard-won expertise in relations with existing code takes place through alignment, trust-building and ongoing negotiations of the conditions the prolonged existence of software. When maintainers step aside from a body of software, whether for reasons of burnout, disinterestement, professional promotions or something else, this knowledge is effectively lost from practice.

Besides bringing practical labour-force to a software project, new contributors were considered as a potential source of the much appreciated, though somewhat mythical "diversity". At these events this term was primarily used in reference to diversity of gender, race and geography. Implicitly these surface-markers serve as proxies for more abstract, deeper and arguably more relevant cultural diversity. While laudable in itself, this proxying runs the risk of masking any number of other social structures, such as class – a middle-class programmer from Iran joining her middle-class programmer peer from France might not "diversify" the documentedly "feminised and racialized low-status labour of maintenance" (Vinsel and Russell 2020) as much as might be hoped for.

In addition to categories of gender, race and geography, less politically contentious diversities of skill, interest and personality were aspired for by those currently already involved in keeping software in order. The personality traits of both introverts and extroverts were argued for being especially valuable in service of maintenance; introverts for their supposed dedication, attention to detail and preference of non-human over human concerns (namely material concerns of software rather than "soft skills" of human concerns) while extroverts were lauded for their capacity to "see the big picture", to communicate, and for "empathy" – valuable for preventing and mitigating conflicts. While this reductionist view of human psychology to two broad classes, and its equally reductionist alternatives such as the Big Five personality traits are highly controversial, they were accepted as heuristic sense-making devices observed in my fieldwork. Whether introverts and extroverts would exist on a scale, or whether some situations would afford a person's introversive or extroversive tendencies were bracketed off. Instead the provisional classification of people into psychological categories were used to organise sense-making and the discourse – an ethnomethod (Garfinkel 1967; 1996). Further diversities, such as "neurodiversity", appeared but they shall be the topic of another text than this one.

Thus, besides a value in itself, diversity of maintainers has utility as a source of information. The logic is that increasing the diversity of maintainers enriches the input of information into the maintenance practices from a widened cultural perspectives. Software which manages to invite, accommodate and importantly to retain a more diverse group of maintainers has, to use the emic terminology more "resilience" in the face of "entropy" which tends to make material things, including code, fall apart. The pursuit for diversity takes a utilitarian turn, and backgrounds diversity as an intrinsic value; maintenance oscillates between a moral order, and a pragmatic matter. At the same time maintenance comes to depend on what exactly is locally considered the causal mechanism which connects diversity to the valuable utility, resilience.

While serving as potential sources of diversity in service of software, the consensus in the observed discourse was that new contributors themselves would benefit from involvement in the maintenance effort; maintenance and care are relational. As I discussed above via the two vignettes, software actively participates in the material-semiotic networks of relations which maintain and care for people. It was argued that newcomers would specifically be positively transformed in the gradual "onboarding", "upskilling" and "schooling" process, accumulating experience and skill – or to put it theoretically with Bourdieu, cultural capital (Bourdieu 1986). This is an apprentice-based model, in which knowledge-how (Pavese 2021) is acquired by participation in a community of practice, and incorporated as tacit knowledge (Polanyi 1958); one participant described this kind of learning from the community to the individual to happen "through osmosis", using an analogue with the chemical process. Cultural capital is gradually embodied by the individual, who can later convert this cultural capital into economic capital in employment opportunities (e.g. as CV or portfolio items, metrified reputation on programming websites such GitHub or StackExchange). Such is the argument. Bourdieu however theorises that social capital is the conditioning factor of converting cultural capital, limiting for whom the conversion will be available and successful (Bourdieu 1986). Conversely, software gets maintained as this embodied cultural capital is converted to labour, or "contribution".

But these relations are risky. In the event(ual event) of maintainers moving on or dropping out, these transformative benefits can be lost. From the perspective of the individual maintainer (assuming they continue to engage in programming practices elsewhere), disengaging means they might need to re-align their skills and practices, ie. the embodied cultural capital, with other bodies of software. If they have become too dependent of the software they maintained earlier, re-alignment might pose problems. From the perspective of the software, loss of information from the assemblage and ongoing practices which maintain it can jeopardise its continued existence if the code has become too dependent on the knowledge of individual maintainers now gone. In the extreme software risks the fate of obsolescence and “abandonware”, a term for software ruins completely deserted by maintainers and void of knowledge how to maintain the code.

The vignette *Maybe one more day per week* highlights how these relationships are negotiated. The relationship between Stefen and the world-building tool has developed gradually, co-constituting both. Over time the tool, and by implication Stefen, has become entangled in other software built by others elsewhere, as they’ve adopter the tool, and weaved it into their code. Those other assemblages are now dependent on Stefen’s biography, and whether he will be able to maintain the tool, and indirectly the scaffold, strengthen and prolong the existence and ongoing becoming of those assemblages. Should Stefen’s and the tool’s relationship weaken, other people and software might need to re-arrange themselves to strengthen the network where people and software depend on one another.

Let's consider "contribution" both as a verb and a noun.

The verb “contribution” signifies the practice of engaging with maintenance of material code. Contribution is not only necessary, but beneficial from the perspective of the entanglement of software and its maintainers, as well as from the perspective of the individual contributor. The word “contribution” is an economic term, tracing its etymology to assigned payments done towards a shared end (“Contribution” n.d.). As such, the word remains mute of the potential beneficial transformation of the one from whom the payment is drawn. At the software maintenance events the word “contribution” carries a strong positive connotation. Besides a verb, “contribution” is also a noun denoting a unit of work. As such, it is a unit of exchange, and a unit of measurement and control. What then counts as contribution, and more specifically what counts as a *valuable contribution* raised some debate. I witnessed a discussion where some maintainers advocated for non-programming tasks to be included as “contributions”. Suggestions included improving graphical user interfaces, answering questions from users, and even managerial work such as task prioritisation, meeting organisation and minute-taking. Platforms on which software work is coordinated, such as GitHub, do not until now measure these kinds of activities, rendering them invisible from an information systems point of view. The “if you can’t measure it, you can’t manage it” management doctrine from Peter Drucker suggests that extending *metrification*, in other words surveillance of flows of contributors and contributions as the rational and preferable course of action (for analysis of trust in numbers see Porter 1995; for discussion about struggles to become visible in management information systems see Bowker and Star 1999 about classifying nursing work).

The noun “a contributor” – the source of “a contribution” – typically refers to an individual person. In open-source software contributors might indeed be volunteering users, so-called “power-users”. Other kinds of entities too are entangled with computer software via contributions; much of open source contributions come from organisations such as companies. Contributions can also come from adjacent or “upstream” software in networks of interdependent software. To exactly whom then should contribution (as a verb or a noun) be attributed to in more-than-human hybrid networks? Clearly not only to humans and their accumulated cultural capital. A more inclusive, more pragmatic approach to analysing the conditions of software is necessary. Let's consider how documentation (likewise both a verb and a noun) adds to how code hangs together and amends viewing humans as containers of information.

To mitigate against risks of information loss, my fieldwork is profilated by calls for better and more extensive documentation of maintenance work. While writing down the sufficient and necessary knowledge for a body of software to continue to exist is a Herculean task and understood to be doomed to never complete, the movement toward making the implicit explicit is however a popular direction.

At an immediate level, the source code documents the software simply through supervenience; in other words the source code strictly defines what the software is and what it does. This idealistic view is problematic, and rejected both by practitioning programmers, system administrators and designers, as well as by critical scholars of software. Beyond formal definition, the process of execution and the socio-material context in which it takes (or does not take) place matters (Chun 2016; c.f. Galloway 2004; Lessig 1999). What even counts as “source code” is a question of debate (Couture 2019). If code is the *text* of software, including its *con*-text is necessary for this documentation effort. Documentation of software has its own idiosyncratic practices gradually developed in software engineering, including human-written and generated code comments, automatic tests, code review (Bialski 2019) and programming paradigms such as “literal programming” (Knuth 1984).

In the discourse observed at the field, repeated calls were made to extend documentation beyond describing the technical con-text to so-called “human factors”. Already existing, variously adopted documentation practices include attributing individual blocks of code their creator, archiving deliberation and decisions about bugs and design choices, and including MAINTAINERS file, a conventional text file serving the dual purpose of listing those who have contributed to the maintenance effort throughout the lifetime of a body of software, and providing technically oriented instruction about how to contribute. In principle these practices are in use in maintenance of open source as well as closed source software, although with widely ranging consistency and variable local practices.

The concepts of *mutual knowledge* and *common knowledge* from the field of Philosophical epistemology, logic and Social Epistemology are useful here. By mutual knowledge we mean that all actors (knowers) know that *p*. By common knowledge we mean a special case of mutual knowledge where each knower additionally knows that others know that *p* (Hendricks and Hansen 2016; Vanderschraaf and Sillari 2021). Applied to a group of software maintainers mutual knowledge might mean, for instance, that each maintainer individually knows that a segment of software should not be modernised because some other system depends on its old-fashioned style (see f.ex. Johnson 2016), while common knowledge would imply that everyone can trust that nobody goes on to modernise the segment, thus breaking it, because are known to be aware of this path-dependent constrain. This kind of contextual and situated knowledge is not expressed in source code alone.

The pursuit of writing down local knowledge into a corpus of what is necessary to know to maintain a piece of software, and expecting that all maintainers are familiar with this corpus, is a tactic to externalise and objectify tacit, local knowledge.

These issues culminated in two controversial genres of documents, namely the MAINTAINERS file and Code of Conduct document.

The MAINTAINERS file came to be contested on one occasion I witnessed an argument between the extremes of including absolutely everyone who have made changes to the codebase ought to be mentioned in alphabetical order without discretion, or whether the document ought to rather be updated to include up-to-date information relevant for identifying who currently and actively maintains what areas of the code. The former was argued for giving credit and rejecting screening of whose contribution counts as “valuable”, while the latter was argued for on the basis of being more accessible and helpful for someone who is seeking to get in contact with a maintainer. The former option might lead to listings of hundreds, even tens of thousands of names for large projects, while the latter opens up a pandora’s box of what counts as a valuable contribution. In the vignette But not everyone is Kubernetes! Uwe’s frustration expresses how such concerns of organising and giving credit to thousands of people are a concern of the few, rather than most of software maintainers who lack the resources of composing even most rudimentary technical documentation, relying instead on their personal memory and familiarity with a body of code and the local styles it expressed in code.

Second controversy about extending documentation to cover “the culture”, “human concerns” and “ethics” was the production of the so-called Code of Conduct. Code of Conduct (CoC) is a genre of normative and prescriptive documents, usually taking the form of a list, which lay out the rules of engagement in a community. Any specific community can write its own Code of Conduct, have it dictated by an owner (such as a maintainer), or adopt a pre-existing one. As a shared, communicated document a Code of Conduct serves as a stable reference point in case of disputes. All participants are expected to be familiar with the code and consent to it. Code of Conduct is thus, theoretically, *common knowledge*. CoCs are not specific to software cultures, but appear in a wide variety of communities including but not limited to companies and corporations, universities, activist groups and makerspaces. Bafflingly, mainstream banks well known for money laundering have them (e.g. Danske Bank) as do imperialist military forces (Israeli Defence Force; USA Military), so the genre of CoC cannot be seen through rose-tinted glasses, but potentially corrupted by so-called *ethicswashing*. However, many informants considered formulating and having a Code of Conduct particularly in open source software projects a best practice, and some argued that not having one is a “bad practice” (emic term). The issue is contested though. The arguments for each software having a CoC were based partially on cautionary tales, and partially on principles. Arguments against CoC included seeing them as distraction from the material and overwhelming programming work at hand, dislike of burdensome bureaucracy, potentially never ending process of creating and maintaining a CoC, and a “we don’t need it here” attitude. The cautionary tales repeated the schema in which the lack of a CoC document allowed disputes to escalate, leading to unfortunate suffering of people, communities and code in toxic contexts which lack authoritative reference point for de-escalation and resolution. But what if an asshole (emic term) makes valuable contributions? It was reported to have happened; while being an asshole excludes being a good maintainer in the wider sense which recognizes the intersubjective work, it is orthogonal for writing useful code. Maintenance is often scarce, so is there opportunity to reject contributions on moral grounds? In the stories, the narrative figure at fault was a central maintainer who abused their power. A CoC would, so it was argued, pre-empt some of these situations. Critics counter-argued that a CoC has no effective power and its mere existence does not turn vicious personalities to virtuous ones, and can distract the maintainers from the material work of keeping software in good working order to “politics”.

While anyone might “act like an asshole” (an emic phrase), the textbook sociological analysis of power states that abuse of structural (rather than idiosyncratic) power requires occupying a powerful subject-position. The shared cautionary tales of maintainers exercising abusive power hints at power-relations inconsistent with the received notion that maintainers rank low in social status hierarchies. The call to objectify prescriptive norms as common knowledge is a collective response not only to individual misconduct (whether through malice or misunderstanding), but to potential abuse of structural power.

While the informants recognized that the attempt to further extend documentation to formalisation and objectification of “culture” to “human issues” using documents such as Code of Conduct will remain open to interpretation and contestation, a consensus was clear that movement towards making “culture” (emic term) more explicit is a worthy attempt for a more clear, accessible and just maintenance of software. Culture was something to clarify and write out; a sentiment any ethnographer (literally "people writing" or "culture writing") might get quite excited about.

Whether argued based on narrative or on principles, the desire for writing down and making visible community rules as common knowledge is, ideologically, consistent with how source code constitutes software. Firstly it makes explicit what shall happen, and secondly serves as a textual “single source of truth” (emic concept, adopted from design) to which all relevant actors may refer to. The first resembles good programming practice, captured in the aphorism collection known as *Zen of Python* as “Explicit is better than implicit” (Peters 2004). The second too resembles how source code defines software, *a-priori* its execution (for critique see Chun 2008; Couture 2019).

5.4 Synthesis

Materializing and objectifying cultural knowledge onto programming tools is a well established practice. To substantiate the claim, materializations include norms such as æsthetic preferences of tabs versus spaces for indentation, *CamelCase* versus *snake_case* for symbol names, or writing documentation for programming functions in imperative tone, to name some examples. While some code is rather formal and abstract more akin to mathematical or musical notation, most programming practice favours semantics and meaning. In addition to prose in style guides and “plain text” (emic term), programming tools such as text editors, style checkers, static code analysis tools (aka linters), integrated development environments (IDEs), compilers are configured or programmed to encourage or enforce not only formal norms but also material-semiotic norms such as commenting.

From an functional ahistorical perspective – one which highlights development of new software, and the function of software outside of its context of production – these kinds æsthetic choices seem trivial beyond collegial and isolated squabbles over what programmers consider fashionable. While this squabbling is without doubt part of “programmer culture”, seen from the perspective of maintenance and long life along with software, the establishment and governance of consistent style becomes a necessity. Non-normative code is hard to read and understand, confidently make modifications to, and to pass on.

Computational norm-governing tools are at work well beyond the present moment of programming, while typing code in a code editor. An illuminating example is *git blame*, a command of a popular version control system to investigate exactly who has made what changes to specific source code at an extremely detailed level. *Git blame* obviously carries strong evaluative norms with its “unfortunately chosen name”, as one informant put it – to consider the normativity it puts forward through its name alone; imagine if the very same command was called “git credit” instead! Similarly unit and integration tests, small secondary programs which continuously check that the primary software works as expected, can come to be contested; when changes to the code leads to tests starting to fail, perhaps the tests themselves had expressed a misguided or outdated idea of the behaviour of the primary software. Such archaeological tools animate the local material archive, open it for normative re-interpretations, and keep the technographies of artefacts and biographies of people bound to one another over time.

All of these tools are computational and algorithmic. They are meta-software; software about software, programming about programming, code about code. As such, the programming tools are not discrete from the main corpus of software, nor the cultures that maintain them, but entirely continuous with them. And like all culture, norms are not made up each time, but are rather reproduced, perpetuated and made durable over time from programmer to programmer, software to software, tool to tool (Latour 1990).

The impetus to expand making the implicit explicit, the discursive material, stems from two sources. Firstly it draws from the understanding that software maintenance communities can be dysfunctional, and the reasoning that making cultural assumptions and preferences explicit is an effective way to pre-empt, de-escalate and resolve disputes and thus protect the community and its members. Secondly it draws from analogous cultural knowledge that software can be dysfunctional, and making assumptions and preferences of its operation explicit can similarly protect the code from bugs, hard to understand “spaghetti code”, and ultimately undermining its continued existence. These two cultural insights have the same logic, and converge in the process of expanding documentation.

This convergence only makes sense against a background of shared experience about what life alongside software is like. The debates at the software maintenance events presumed that such a first-hand experience – the experience of software maintenance was considered common knowledge to base arguments on, and a necessary and sufficient background against which the stories were relatable and the arguments agreeable.

A shared maintainer experience provides the third and underlying impetus for making explicit the implicit knowledge necessary for keeping software in decent working order. At the core of this experience is the sense of insufficiency in the face of the contemporary artifactual world. The experience of software maintenance is a subjective reflection on the materiality, practicality and relationality of dependence on digital infrastructures ones does not quite rely on. Maintainers recognize one another as maintainers, and it is against this shared, reflexive experience that the stories are understandable and evocative. The experience is a reflection on the fragility of personal and shared life within these architectures, and trying to care for what can be cared for. It is an experience of a modernised, globalised out-of-control, decaying *runaway world* (Giddens 1984; Bryant and Jary 2003), and the desperate attempt to try to write everything down for possible future use is *broken-world thinking* (Jackson 2014).

While shared, the experience is varied and multiple, as exemplified in vignette *But not everyone is Kubernetes!* The experience was stable enough for Uwe to recognize the argument and appreciate the work done at the major, company-driven open-source software project, but he saw no avenue to translate it into practice within the more marginal software project he maintains. More generally, translating information from one site of maintenance to another one is hard. While advice might be interesting, keeping a body of code in good working order will always need to be done locally, in the particular. It reminds us that software is not a *kind*, but a rhizome of highly diverse materials and peoples in various states of decay – *a broken world* (Jackson 2014). Decay can be catastrophic, a breakdown; at one of the events of my fieldwork the very first session to be proposed was straightforwardly titled “avoiding burnout”. It overflowed with participation.

The vignette *Maybe one more day per week* foregrounds the shifting agencies and affects software maintainers identify during their personal biographies and the biographies of technological artefacts, and how the torque of these life-forces is conditioned by further bundles of relations. For example, various game developer’s relationships with their games depend, partially, on Stefen’s ongoing relationship with his employer. Such relationality is well studied in infrastructure studies (Star 1999) and in maintenance studies extending this tradition (Denis and Pontille 2019). What detailed analysis of software maintenance contributes is nuance into the varied and shifting positions within which maintenance of technological infrastructure takes place. Stefen was in a position to hesitate, and to reflect on this affect. In contrast to analyses of subject positions of maintenance workers generally, many subject-positions of maintainers are positions of power – even dominant and sovereign power as made obvious by the calls to formulate Code of Conduct documents to restrain possible abuse of this maintenance power and protect psychology of the maintainers in addition to the materiality of the artefact.

In contemporary software maintenance the centrality of tools is apparent, and describe and prescribe how it is conducted. Tools serve as crutches for newcomers and seniors alike. A key part of onboarding newcomers to development and maintenance is instruction and disciplining on correct application of the preferred choice of tools. To be able to bring and sustain people in these relations, the tools must be accessible. Accessibility means availability and being free of cost, explicit aims of the free and open source software (FLOSS) movement. Additionally tools must have comprehensive documentation and tutorials, a good "developer experience (DX)" (emic term from my fieldwork) and importantly an active community of relevant users.

Some of the tools are online platforms, GitHub is the main one of these. How practices are captured by the tools matters – in my fieldwork I am sometimes not sure if the informants are describing their own values and practices, or if they are describing GitHub products. For example on one occasion I needed to ask an informant speaking about "organisations" to clarify whether they meant "organisation" as a company, NGO or other group of people gathered around common goals and resources "in the real world", or "organisation" as a set of GitHub features and I was met with a blunt answer: "the GitHub features". That's how fused they are.

Open source software, an infrastructure on which the entire digital realm along with it's own tooling has come to depend on, is maintained in the public. This exposes not only the source code, but also the developers and maintainers to feedback, feature requests, bug reports, code contributions, grievances and questions from users. While this openness is a foundational core value, visibility and transparency has its downsides when operating at scale. A software project, and indirectly its maintainers, might receive any range of attention and the ease with which users, other programmers and other publics are able to engage with a software project through platforms like GitHub was the topic of active debate and a variety of stories. For Uwe a sustained interest in his world-making tool and its incorporation into other software enabled new opportunity, while for Stefen the failure to onboard peers meant he felt unable to face the accumulated maintenance burden.

The openness which enables "random people on the internet dumping their problems on your todo list" is a controversial and recurring theme at the events, and studied in depth by Geiger, Howard and Irani (2021). The session on burnout overflowed with participation, suggesting that the exhausted maintainers had been used, used up (Ahmed 2019). "Did I sign up *for this*?" they asked one another. The "culture of openness" has put too much attention on the rights and abilities of the users, not enough on the maintainers of digital infrastructure, some informants argued.

5.5 The relation of *maintain-ability* hinges materially on practical capacity-to-maintain-and-be-maintained

In the research field of material culture, culture is understood as participation in production, use and exchange of meaningful objects – *things* (Ingold 2007; 2010) –

and passing on of practical knowledge on how to live and flourish among them. Sharing first hand experiences, ie. culture, thus has a material basis (Latour 1990) and echoes Marxist historical materialism. Inter-subjectivity and recognition of one another depends on this material and material-semiotic basis. Software is the ecology of *things* among which my informants spend their time and their attention, and which they depend on for identity, relations and subsistence.

According to the ecological view of skill, skill is not a property of an individual, but instead a relation between the individuals and *things* (Ingold 2007; 2010). “The study of skill demands a perspective which situates the practitioner, right from the start, in the context of an active engagement with the constituents of his or her surroundings” (Ingold 2000, 5). This *dwelling perspective* (Ingold 2000, 5) was inspired by work of process philosophies of Bergson, Deleuze and Guattari and others which abandon classical metaphysical analyses of the world into objects and their properties, and (re)introduce a continuous, rhizomatic analysis into materials and forces acting on it. (Ingold 2010).

I name the relation my informants debate *maintain-ability*.

The concept of *maintain-ability* references the notion of *response-ability*, the capacity to respond and be responded to, introduced by the influential feminist scholar of science and technology Donna Haraway. Haraway discusses response-ability as a necessary capacity for *staying with the trouble*, and to live and die on a damaged planet (Haraway 2016). The notion has further been advanced by other feminist scholars of technoscience such as Karen Barad (Barad 2007) and Maria Puig de la Bellacasa who develops a feminist theory of care from Haraway, the Philosophical lineage of ethics of care from Tronto (Tronto 1993) and New Materialist work of Bruno Latour (Puig de la Bellacasa 2017). Response-ability is the *ability-to-respond/ability-to-be-responded-to*, and it’s care extension the *ability -to-care-of/ability-to-be-cared-for*.

To this, *maintain-ability* provides important qualifications. Literally, *maintain-ability* obviously pays attention especially to maintenance, a co-constitutive and always situated set of practices of living with meaningful *things*, similar to Ingold’s dwelling perspective (Ingold 2000, 5). Substantially *maintain-ability* keeps in view the local and always material microhistory of which it is an outcome of. In Tim Ingold’s terms, the relationship of maintainability is a line rather than a point – a process rather a object (Ingold 2010). As a relation between material and forces it is not a skill the maintainers possess, or a property of well written code, but the relational *capacity-to-maintain-and-be-maintained*.

Similarly to response-ability, my concept of *maintain-ability* is a conditional and vulnerable relation which can fail. Like “sustainability” being a virtual potential, a fragile possibility of actually being sustained. For example, environmental sustain*ability* does in no way guarantee that any particular environment will finally be sustain*ed*; in Philosophical concept analysis sustainability is a necessary, but not sufficient condition. Analogously the property of *maintain-ability* is a hopeful potential to maintain-and-be-maintained. Maintain-ability backgrounds both the maintainer Subject acting on an Object world, as well as the material which depends on maintenance as an Object. Maintain-ability is transitive and diachronically passed on through material and time in the sense that the conditions of continued existence of future *technological a-priori* (Kittler 1999; Tuschling 2016) are enacted and re-enacted today.

Maintain-ability denotes at once the subjective, lived practical experience of living alongside and *dwelling* in computer software, the material basis for inducing inter-subjectivity between tool-users, and the relational attempt to transcribe mutual knowledge onto artefacts as common knowledge.

This concept is the main finding of my thesis, and concludes this chapter.

6 LESSONS FOR MAINTAIN-ABLE WORLDS

How does the concept of *maintain-ability* developed in the previous chapter in the context of computer software contribute to the study of science and technology, and more generally to life in a material world? In this chapter I will sketch out some applications of my finding and weave it with published literature.

First an incomplete list of caveats.

My fieldwork is based on participant observation and unstructured interviews at four meetings which partially overlap in their theme. The unifying theme is, of course, maintenance. All the events explicitly had the word “maintenance” in their name. The concept therefore served also as my sampling strategy for attending exactly these events, and not others, and this bears some implications for what can be learned from this research. Without a doubt much of discussion about maintenance takes place in situations which are not explicitly marked to be about maintenance. According to Rebecca Mossop (informal communication at REPAIR project presentation in January 2020), the word “maintenance” is not necessarily the emic term of choice, and does not in fact appear very often in the relevant discourse. For instance the everyday vernacular language about programming prefers expressions like “fixing bugs” or simply “fixing”, “refactoring”, “debugging”, “redesigning”, “reviewing” or “rewriting” for dealing with the material code. Other activity around the core practices of touching code might be expressed as “opening” and “closing issues”, “documenting”, “posting” on mailing lists or discussion forums, “suggesting” alternatives, “replying” to requests, “fighting” about opinions, “raising” and “collecting” funds, “burning out” and “stepping down”. Software maintenance constitutes a mix of all these activities, and many more, under the umbrella activity of “programming” (Mackenzie 2006). Programming is a site of rich language use (even more so beyond the English language above). Following the exact term “maintenance” is therefore a commitment to a specific concept whose stability is an achievement, a partial outcome of the events studied. “Maintenance” is a *lateral concept* (Gad and Jensen 2016).

Secondly the events were based on discussions – no actual, “real” maintenance got done during these events, and I did not see a single line of code. It might be argued that these events even distracted from the material maintenance activities in the codebase. I consider this interpretation shallow and insufficient as too idealistic, divorced from the local, material histories and lived experience alongside that code, and and too focused on programming; while no bugs were fixed or diagrams cleaned during this time, the relations and practices were strengthened, documentation written and the first-hand experience of maintaining software recognized and affirmed – what I call *maintain-ability*. I too struggle with this lack of material encounter in my own research, and the propensity of the discussions to not speak of code or programming practices feels unsatisfactory to me. "When do we finally start to talk about that real thing?" – referring programming – was a thought, a lack, *manque* (Lacan 2006) I had to learn to let go of during the fieldwork (Ojala 2020). Personally, as a media theory and design trained science and technology studies scholar committed to the project of bringing the material back to social enquiry, this was admittedly a bitter pill to swallow. The present absence of code is, however, consistent with ethnographic research which has attended meetings of programmers and others who live alongside software such as scientists and engineers (Cohn 2019; Bialski 2020), and “work ethnography” more generally (Orr 1996; 2006). The meetings and the discourses are in some way “the residue” of concerns when people step back from the material object of/in their case. Experienced programmers are by no means surprised by this, nor are science and technology studies scholars in the ethnomethodological or Actor-Network Theory (ANT) traditions who take it for granted that the informants themselves are themselves competent sociological, cultural and political theorists whose “folk theories” must be included in the analysis (Callon 1984). However caution must be exercised whenever practices are not studied empirically in “natural settings” through f.ex. ethnography or other relevant methods such as (re-)enactment and first-hand experience acquisition techniques of media archaeology (Ernst and Parikka 2012; Piccini 2015) or practice based research. In summary, meetings of software maintainers are very limited ethnographic sites, even for multi-sited ethnography (Marcus 1995).

Thirdly, the attendance of these events was obviously biased. Besides the desk research and personal experience, my fieldwork hinges on the several dozen informants who had the chance to, and chose to attend the meetings I did too – in this place at this time. What do global entanglements of software maintenance look like in, say, India and in the COBOL language (Ritasdatter 2020)? What were the debates of software maintenance before GitHub (Zhou, Vasilescu, and Kästner 2020), or before Internet access? What will be the issues of maintaining code generated through machine learning techniques (Mackenzie 2013; 2017)? What are the experiences and scalar labour necessary for open source software to grow (Geiger, Howard, and Irani 2021)? Beyond geography and time, the voices who would wish to distance themselves from software maintenance for reasons of ignorance, disrespect, personal tragedy or dishonesty would by research design be absent from my fieldwork. And of course the gender-bias issues of “tech” are by now well documented (Plant 1997; Ensmenger 2010b; 2010a; Hicks 2017). While I theorise, my empirical work does not intend to generalise, and generalisation is not a necessary property for qualitative, ethnographic research methods whose aim is not statistical representation but fuller, *thick* (Geertz 1973) and *broad* description which includes more of the margins (Becker 1996).

These caveats considered (and the list could go on), lessons learned from software maintenance do carry to life with artefacts more widely, some of them posing challenges to the published literature. Firstly, software maintenance is done, according to my fieldwork, mostly by men. This raises questions concerning the analysis that maintenance typically falls on the shoulders of women. Other technologies are typically maintained by men too (e.g. Nova and Bloch 2020; de Wilde 2021), lending to trashy stereotypes of macho construction site workers and car mechanics as well. It seems at the same time to be true that maintenance is feminised and technology masculinized. This analysis is an epiphenomenon of a limited, cherry-picking and unfortunate understanding of what “technology” is considered to be, and what the object of maintenance is. As pointed out by Bowker (1994), Star (1999) and Bowker and Star (1999), established technology tends to vanish out of sight as background infrastructure Star (1999) as it is domesticated and becomes “mundane”. Impacts of technology in the home were originally studied by women’s historians such as Cowan (1985). Seen in connection with the explicitly affectionate notion of care and it’s emergence as a research topic in Science and Technology Studies (Mol, Moser, and Pols 2010; Lindén and Lydahl 2021) and beyond maintenance, technology and together maintenance of technology are all complex nexuses of figures and attributed meanings. Technology maintenance is – in the light of masculine/feminine binary for gender of people or more cosmologically – *queer*.

The concept of *maintain-ability* is helpful here. Unlike essentialist philosophy focusing on substances and their properties, the relational concept of *maintain-ability* names the enabling condition of being, flourishing or at least surviving in a broken world. As I’ve theorised my fieldwork based on Ingold’s ecological view of skill (Ingold 2000) and Haraway’s *response-ability* (Haraway 2016), the relation depends on capacity-to-be-maintained and capacity-to-maintain. Where can we find this capacity to be thinly distributed and weak, and where richly distributed and robust?

A poster technology whose capacity to be maintained is strategically hindered is the Apple iPhone (Vinsel and Russell 2020). Other well documented, specific cases are the John Deere tractors (Saariketo and Glöss 2020). In the prolific case of the smartphone the weak (and weakened) maintain-ability is related more to its physical aspects such as the glass screen and the battery rather than the software it runs. In the cases of farm equipment and automobiles the controversies are about their computerization which establishes explicit barriers and requires any repair or maintenance to be done within the governance regimes of the manufacturer, surveilled and disciplined through mechanisms such as certifications and proprietary standards and tools.

Maintainability of software is hindered if source code is kept unavailable. Such “closed source software”, an antonymic and pejorative term coined by the proponents of various open source ideologies, is a power struggle over who the relevant and appropriate maintainers are considered to be. The right to study and modify software source code is one of the basic freedoms open source lays out. Closed source blocks this access, and fundamentally denies whether access to code is a right. Another strategy of hindering software maintain-ability are the various techniques of code obfuscation by computationally making the code very, very hard to read and intervene in while retaining its functional logic. An example of this is JavaScript code on websites. The executing code itself is open source by the design of the infrastructure of running it on the client side, but mangled in ways that make it practically impossible to understand.

Besides explicit barriers of repair and maintenance (such as glue on an iPhone screen, security measures of a John Deere tractor CAN bus, or rejecting access to source code), relations of maintain-ability are thinned by dis-interessement, de-skilling and alienation. Consumers placed in the subject-position of “the user”, or “the end-user” (both emic, key concepts also in software) expect their technology to be “user-friendly” and “usable”. In a position of “the user”, one does not expect to be interested in establishing or developing a meaningful relationship to the tool at hand, an ideology which is nourished by (unfortunate) ideas of technology being a-personal (see Ahmed 2019 for wonderful meditations on the word “use”). Rather, is is the task which has primacy, and the task may be maintained by simply replacing the tools when necessary. A user’s intention reaches beyond the tool which is reduced to being *zuhanden*, ready-to-hand to use Heidegger’s image of the hammer (Heidegger 1977). Being dis-interested in the tool itself is a virtue of the figure of the User.

An unfortunate process through which dis-interessement takes place is for some technology to start to be considered “legacy”. Ritasdattir has studied how the COBOL programming language has fallen into the category of legacy, while at the same time being one of the most important languages in which key infrastructure runs on in e.g. banks and other finance institutions. Maintenance of COBOL code has been outsourced to India, where it has been received with great optimism for participation in the global economy through this maintenance work. (Ritasdatter 2020). A more micro-level study of a programming language falling into legacy is given by Cohn, describing how a maintainer of a program written in tcl/tk was struggling with the pressures of at the time modern Java language (Cohn 2019). This lineage continues in the illustrative vignette *But not everyone is Kubernetes*, as Uwe faces struggles to interest others to participate in software written in Java, now sometimes considered a legacy language itself. A troubling reflection: Java is the language I was taught in my own computer science studies, and we teach it to computer science students at IT University of Copenhagen. Programming languages and paradigms fall into legacy, and sometimes even re-emerge from it in the political economy of software.

At one of my fieldsites, Festival of Maintenance, textile designer Tom van Deijnen alias Tom of Holland presented a campaign for “visible mending”, ie. maintaining textiles in a fashion which not only leaves visible the patches and repairs, but celebrates them, a kind of a soft *kintsugi*. According to their research of textile craft and domestic handbooks, instructions of repair have historically waned and all but disappeared from housekeeping and crafts books (van Deijnen 2019). At the same event and similarly pointing to thinning of *maintain-ability*, the vanishing of schematics which used to accompany electronics such as televisions, radios, musical instruments, amplifiers and the like was highlighted by “right to repair” campaigners.

In theories of tacit knowledge (Polanyi 1958) as well as in Ingold’s ecological view of skill, the fluency to improvise in the material world is acquired through *dwelling* (Ingold 2000). The capacity to repair and maintain is developed by gathering sensual knowledge (Dant 2010). *Capacity-to-maintain* too is acquired not through documents or formal education, but through practice over time with materials which have the *capacity-to-be-maintained*. In co-constitutive relations of maintain-ability the subject can transcend the narrow and incapacitating position of the User, and discover meaningful positions for themselves within the material world. Dis-interessement, the undoing of Callon’s notion of *interessement* for strengthening the ties between actors (Callon 1984), alienates the users not only from their material world, but from one another when they are deprived of for instance sharing with one another the practical, embodied skills of living in a material world. To use the example of a computer system which does not support maintain-ability, users do not support one another by sharing tips, “hacks”, frustrations and desires, but instead are faced toward the producer of the technology individually, in alienation. While the user is skilled in the *use* of the technology, they are unskilled (deskilled) in its maintenance, thinning the *maintain-ability* relations.

Things are of course more nuanced than a sketch for a theory of deskilling under the conditions of late capitalism would let us believe. People living in the material world share many kinds of advice with one another about how to do it successfully – repair and maintenance documentation and tutorials are collected on specialised websites such as iFixit, and on YouTube. Some of these practices have organized as movements such as the Right to Repair movement calling for refurbishing, reusing and repairing of electronics such as smartphones, computers and printers (Vinsel and Russell 2020). Not surprisingly Right to Repair was visible at my fieldsites both as being talked about, as well as Right to Repair activists participating in the meetings about maintenance. The movement has been moderately successful in lobbying for legal rights in the European Union, United Kingdom and beyond, pushing product design and manufacture toward what I conceptualise *maintain-ability*.

The right to repair sits at the core of free/libre open source software (FLOSS), and is codified as the right to study and modify the source code. While “Right to Repair” makes good sense for a programme to advocate change in the legal register, the name points to some issues. Very generally the idea of “rights”, whether natural, legal or universal, is a contested idea in political Philosophy. More specifically Nandita Badami has critiqued repurposing informal repair and the “right to repair” as a “fix” for the global economy (Badami 2018; Reeves-Evison and Rainey 2018). While rights, such as those advocated by Right to Repair, are of course important, it is also crucial to think about the conditions within which such rights are exercised and concretized in the material world; care is about practices not principles (Mol, Moser, and Pols 2010). Thinking in terms of capacities can ground idealistic rights in materiality, practices and relations. The aim of the right to repair is not necessarily to expect the consumer to mend their own devices individually, but to have access to a functioning secondary market of legal repair services. Work and knowledge practices in sometimes borderline gray market mobile repair shops is ethnographically documented in Nova and Bloch (Nova and Bloch 2020).

Maintenance of software has its idiosyncrasies. Software depends on source code, the written description of what the software is and does. While what counts as software is contested (Couture 2019), and it is known in academic research (Chun 2008) as well as through practice that the code is not enough, it is however – in one form or another – a necessity for the existence of software. Access to code is crucial for its maintenance, not so much in the strict sense because software supervenes on code in an ontological sense, but for practical reasons – the programming languages in which code is written are purpose-made for reading, writing, organising and discussing by humans. The history of programming languages is that of scaffolding individuals and groups of people to materialise more abstract operations in machines (Mackenzie 2006; Kelty 2008; Chun 2016; Mackenzie 2017; Marino 2020), often designed to resemble "natural languages" such as English in the case of COBOL and BASIC (Ritasdatter 2020; Marino 2020), and Cree in the case of Cree# (Marino 2020; Corbett 2021).

One very specific reaction to the centrality of source code is, of course, open source software (OSS). This freedom and openness itself is an outcome of an ongoing global struggle, and this thesis does not aim to provide a history or cultural theory of open source, or summarise the various shapes free, libre and open source software (FLOSS) has taken since 1980s and 1990s (see e.g. Kelty 2008) and will take in the future. Suffice to say that much of FLOSS maintenance takes place within a wider context of life alongside software, as we've seen in this thesis (see also Geiger, Howard, and Irani 2021). Besides the empirical research I've presented, the introduction anecdotally and autobiographically demonstrates some of the complications of materially, practically and relationally maintaining software: while the modest WordPress plugin I have programmed is free and open source software made available in the WordPress Plugin Directory and it's code as open source on GitHub, I cannot help my colleague Kasper to fix the website of their research group. Much more than access to source code and labour force is needed for *maintain-ability*.

As perhaps the most supposedly "immaterial", or ideal of technologies, software too requires maintenance. Software is not, as either naïve, tragicomic or malevolent "tech ideology" would have us believe, immaterial and without history and legacy (Bialski 2020). It exists *longue durée* and is reproduced over time in complex, improvised, fragile and historical hybrid entanglements. It is within these entanglements that all biographies and technographies pan out. No software is a new thing of today or the soon-to-come exciting near future, but part of the technological *a-priori* and merges into its legacy. Or to put it more dramatically, part of the piling wreckage we perceive as a chain of events and call progress (Benjamin 1940).

7 CONCLUSION

Research on care, repair and maintenance has focused on the invisible, backgrounded, poorly recognized and suppressed aspects of these low status activities. Arguments for paying more attention to it have focused on one hand on the strict material necessity of maintaining the world we occupy from various economic and managerial perspectives. Some arguments prioritise existential issues of the ongoing climate crisis and toil under late stage capitalism. Social justice critiques prioritise the deepening global inequalities of race, gender and class and how they play out locally. Moral arguments instead valorize the small and poetic, intimate, modest, personal, romantic and even bourgeois – the often used example the japanese art of *kintsugi*, ie. repairing broken pottery with gold being a key example or precious repair. These are all related research perspectives, very necessary, and my own research draws in part from all of these.

All research, perhaps, hopes to uncover something which was previously unseen. Critical social science research further takes on its agenda to reveal the underlying mechanisms and epistemological politics which make some topics poorly visible, others highly visible, and describe and remedy the injustices caused by biases of research attention. Study of infrastructure has been one of the areas where this has been productively done in regards to technology, foregrounding what had receded into the background, while backgrounding what was screaming in the foreground – a research move Bowker and Star call *infrastructural inversion* (Bowker 1994; Bowker and Star 1999). Recent critiques of digital infrastructures have focused particularly on the data and algorithms that make up these digital infrastructures as a background of public and private life, foregrounding the many grave exploitations and unintended consequences (e.g. Ensmenger 2010b; 2010a; O'Neil 2016; Noble 2018; Zuboff 2019). While balancing on one hand description, and on the other the need to render something urgent, some research of digital infrastructures and hidden technological dynamics tends to edge on *algorithmic drama* (Ziewitz 2016) and even hyperbole. I hope not to be misunderstood here – drama is a valuable genre of poetics and one of the many that has a place in research and in the stories that tell stories (Haraway 2016), a productive literary technique for analysis, as well as very entertaining.

Denis and Pontille channel Star's critique that the classical, Actor-Network Theory (ANT) tradition of science and technology studies (STS) has tended to side with the winners, foregrounding science and technology which is visible – the megaprojects, familiar brands and achievements. Denis and Pontille remind us of her program of ethnography of infrastructure (Denis and Pontille 2019), or what came to be known as "infrastructure studies" would help focus on all of which those aspects of socio-technical systems which have been backgrounded. And indeed, various infrastructural inversions have critiqued algorithms of Facebook, Amazon, Apple, Microsoft, and Google (FAAMG) or whatever the handful of US American big tech megacorps everyone loves to hate at a given time.

Focusing on these megacorps and unearthing the social wreckage they leave in their wake does not however live up to the hopes for the project of ethnography of infrastructure (Star 1999). Instead, critique has become hypnotised and been co-opted. Single-minded focus on publicly visible mainstream megacorps further validates and reifies the powerful positions of those they aim to critique. This is most obviously tangible when we observe the units of critical analysis: the corporations themselves are often considered to be the most relevant entities to critique, and are granted powerful agency. While it is a trivial, indeed tautological claim to say that the dominant actors have become to be the most relevant entities to critique, I find this situation to cause shortsightedness in ontological politics of critical study of science and technology, and more relevantly for my own research projects, study of computer software. Software is implied to be – by the most concerned – a technique for megacorps to use for the purposes of capitalist exploitation, and governments (in alliance with the said megacorps) to control their populations. I challenge the ontological import implicit in accepting branded megacorps or products as relevant units of analysis.

I have no objections to the analysis that computer software is used in capitalist exploitation and state suppression. However, such a view of computer software is extremely narrow, false and has the unfortunate side-effect of masking many interesting areas of software culture. Advocating for ethnography of work, Orr argues that abstractions of work (Orr 2006), while powerful, do not necessarily help improve analysis or help critique to make helpful interventions in the world. They delete the work and ultimately serve only the management. (Orr 2006). I add that focusing on branded megacorps or products such as algorithms serve their PR departments and shareholders. I have aimed to provide complementary analysis by thickening the description of life alongside computer software. Through the perspective of broken world thinking, my attempt has been to make visible again the lived experience in a world where software is an existing fact, rather than an existential threat. I have provided nuance by describing biographies of software, and contextualise the lifeworlds in which those lives are lived. Rather than concluding once again that various kinds of imbecile or nefarious technohype works to the benefit of late stage capitalism (it does!), my project took this fact as a point of departure, and then tried to describe some of the phenomena certain subjects of the Capitalocene, those engaged in maintenance of software, materially, practically and relationally respond to it in the biographies within which they find themselves.

In this thesis I have systematically backgrounded the usual actors of studies of digital technologies, namely the corporations. They did not come up in my research, and I have little theoretical interest in them. Instead I have tried to follow the calling of anthropology (the revised, no-longer-colonialist kind) and ethnomethodology, and allow the fieldwork, material and the informants speak to me to build a thesis and a critique of technology maintenance, and of software.

The fact that the proposed sessions about burnout in software maintenance overflowed with participation signals a troubling phenomena. The participants actively recognize burnout to be a widely shared concern. This invites us all to ask what holds together the relations on which our digital infrastructures depend on. To reflect on our positionality within these infrastructures, I suggest we consider Maria Puig de la Bellacasa’s note: “the soil you depend on depends on those who depend on you" (2017, 177). We depend on the morals, ideology, working pride, everyday necessities, biographies and identities of those programmers and their material, tooled-up and torqued relationships alongside computer software.

Through development of the theoretical concept of *maintain-ability* this thesis frames the relations as fragile and vulnerable, and enquires into what does it take to be able-to-maintain, and able-to-be-maintained. Both the moral and practical work of maintenance is demanding as it is rewarding. The costs of failure are real, and fall on both software and people alike, with repercussions vibrating along the material or rather material-semiotic relations which holds life together and gives it a temporal basis (Latour 1990). According to my informants the incentives, rewards, resources, expectations and vulnerabilities are out of balance. It is through-and-through political who has the ability to participate in maintenance of software, and whose life and livelihood it maintains, what knowledge counts as which standspoints are epistemologically valid (Denis and Pontille 2019), who are considered valuable contributors, which software has the capacity to be maintained, and which software projects fall by the wayside. As documented in literature on history and sociology of technology, this maintenance is usually backgrounded and made invisible. These curtains are drawn for instance by labelling some people as "users" (Ahmed 2019), a round-about way of saying "non-participants" and by labelling some software as "stable" or "legacy". Similarly, what it takes to be *maintain-able*, to be able to receive care, recognizes interdependence and the *logic of care* (Mol 2008; Gorm and Shklovski 2019) which holds together more-than-human worlds. These questions are heard in campaigns for open source, Right to Repair, and above in the vignette of Uwe.

It took a "scrupulous detour through the empirical" (Latour 2007; Ang 2011; Winthereik 2019) to arrive at the concept of *maintain-ability*. Fieldwork using the methods of participant observation and unstructured interview proved to be foundational for me to meet the relevant people, my informants, and hear themselves narrate their own struggles, and interpret the experiences and stories they shared. Reflecting on my own entanglements as a software maintainer was critical for understanding what they were willing to share with one another and with me. For the sake of my contribution, as well as for my own sake I have tried to construct a thesis which is not incongruent with my own experience as a (very modest) maintainer of software. The importance of preparatory desk research cannot be highlighted enough though, and desk research was absolutely necessary for me to set myself to be surprised during the fieldwork and in the course of writing the thesis in your hands now.

I finally return to that email from Kasper I started with. I have now come to better understand that the relationship of *maintain-ability* between me and the Pure Feed Widget has become thinned when the code was forked, and the forked code installed for reasons that were relevant at that time. The fact that Kasper contacted me to attend to software of which I am not in fact the maintainer of (or am I? Should I? Could I? What would it take? *Cui bono*?) is a testimony to the complex and interesting socio-technical relations software participates in. I, like you dear reader, have a practical role to play in keeping the more-than-human world going for our own sakes, and for the sakes of others.

This is my thesis.

8 REFERENCES

Agre, Philip. 1997. “Toward a Critical Technical Practice: Lessons Learned in Trying to Reform AI.” In *Bridging the Great Divide: Social Science, Technical Systems, and Cooperative Work*, edited by Geof Bowker, Les Gasser, Susan Leigh Star, and Bill Turner.

Ahmed, Sarah. 2019. *What's the Use? On the Uses of Use*. Duke University Press.

Ang, Ien. 2011. “Navigating Complexity: From Cultural Critique to Cultural Intelligence.” *Continuum* 25 (6): 779–94. https://doi.org/10.1080/10304312.2011.617873.

Badami, Nandita. 2018. “Informality as Fix. Repurposing Jugaad in the Post-Crisis Economy.” *Third Text* 32 (1): 46–54. https://doi.org/10.1080/09528822.2018.1442190.

Balsamo, Anne. 2011. “Designing Culture: The Technological Imagination at Work,” June. https://doi.org/10.1215/9780822392149.

Barad, Karen. 2003. “Posthumanist Performativity: Toward an Understanding of How Matter Comes to Matter.” *Signs: Journal of Women in Culture and Society* 28 (3): 801–31.

———. 2007. *Meeting the Universe Halfway: Quantum Physics and the Entanglement of Matter and Meaning*. Duke University Press.

———. 2014. “Diffracting Diffraction: Cutting Together-Apart.” *Parallax* 20 (3): 168–87. https://doi.org/10.1080/13534645.2014.927623.

Becker, Howard S. 1996. “The Epistemology of Qualitative Research.” In *Ethnography and Human Development: Context and Meaning in Social Inquiry*, edited by R. Jessor, A. Colby, and A. Shweder, 53–71. The John D. and Catherine T. MacArthur Foundation Series on Mental Health and Development. Chicago, IL, US: The University of Chicago Press.

Benjamin, Walter. 1940. “On the Concept of History.” 1940.

Benzer, Matthias, and Kate Reed. 2021. *Donna Haraway: New Modes of Sociality*. SAGE Publications.

Berg, Bruce. 2007. *Qualitative Research Methods for the Social Sciences*. Sixth Edition. Pearson.

Bialski, Paula. 2019. “Code Review as Communication: The Case of Corporate Software Developers.” In *Communication*, edited by Paula Bialski, Finn Brunton, and Mercedes Bunz, 93–111. In Search of Media 3. Lüneburg: meson press.

———. 2020. “Speeding up, Slowing down, Breaking down: An Ethnography of Software-Driven Mobility.” *Mobilities* 15 (5): 740–55. https://doi.org/10.1080/17450101.2020.1816035.

Bogusz, Tanja. 2018. “From Crisis to Experiment: Bourdieu and Dewey on Research Practice and Cooperation.” In *Questions of Practice in Philosophy and Social Theory*, edited by Anders Buch and Theodore Schatzki, 157–75. Routledge.

Bourdieu, Pierre. 1986. “The Forms of Capital.” In *Handbook of Theory and Research for the Sociology of Education*, edited by J. Richarson, 241–58. Greenwood.

Bowker, Geoffrey. 1994. *Science on the Run*. MIT Press.

Bowker, Geoffrey, and Susan Leigh Star. 1999. *Sorting Things Out: Classification and Its Consequences*. Inside Technology. Cambridge, MA, USA: MIT Press.

———. 2000. *Sorting Things Out: Classification and Its Consequences*. MIT Press.

Brown, Barry, Julian Bleecker, Marco D’Adamo, Pedro Ferreira, Joakim Formo, Mareike Glöss, Maria Holm, et al. 2016. “The IKEA Catalogue: Design Fiction in Academic and Industrial Collaborations.” In *Proceedings of the 19th International Conference on Supporting Group Work*, 335–44. GROUP ’16. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2957276.2957298.

Bryant, Christopher, and David Jary. 2003. “Anthony Giddens.” In *The Blackwell Companion to Major Contemporary Social Theorists*, edited by George Ritzer, 2nd ed., 247–73. Blackwell Publishing.

Bucher, Taina. 2012. “Programmed Sociality: A Software Studies Perspective on Social Networking Sites.” PhD Thesis, Oslo: University of Oslo.

———. 2018. *If...Then: Algorithmic Power and Politics*. Oxford Studies in Digital Politics. New York: Oxford University Press.

Butler, Judith. 2001. “Giving an Account of Oneself.” *Diacritics* 31 (4): 22–40.

Callon, Michel. 1984. “Some Elements of a Sociology of Translation: Domestication of the Scallops and the Fishermen of St Brieuc Bay.” *The Sociological Review* 32 (1_suppl): 196–233. https://doi.org/10.1111/j.1467-954X.1984.tb00113.x.

Chun, Wendy Hui Kyong. 2008. “On ‘Sourcery,’ or Code as Fetish.” *Configurations: A Journal of Literature, Science, and Technology* 16 (3): 299–324. https://doi.org/10.1353/con.0.0064.

———. 2016. *Updating to Remain the Same: Habitual New Media*. Cambridge, MA, USA: MIT Press.

Clarke, Adele E., Carrie Friese, and Rachel S. Washburn. 2005. *Situational Analysis: Grounded Theory After the Interpretive Turn*. SAGE Publications.

Clifford, James, and George E. Marcus, eds. 2010. *Writing Culture: The Poetics and Politics of Ethnography, 25th Anniversary Edition*. 2nd ed.

Cohn, Marisa Leavitt. 2014. “Dynamic Reconfiguration in Planetary Exploration - IT-University of Copenhagen.” *M I S Quarterly* 38 (3): 831–48.

———. 2016. “Convivial Decay: Entangled Lifetimes in a Geriatric Infrastructure.” In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, 1511–23. San Francisco California USA: ACM. https://doi.org/10.1145/2818048.2820077.

———. 2019. “Keeping Software Present: Software as a Timely Object for STS Studies of the Digital.” In *Keeping Software Present: Software as a Timely Object for STS Studies of the Digital*, 423–46. Princeton University Press. https://doi.org/10.1515/9780691190600-029.

———. 2021. “Unruly Bodies of Code in Time.” In *Media Infrastructures and the Politics of Digital Time. Essays on Hardwired Temporalities*, edited by Axel Volmar and Kyle Stine. Amsterdam University Press.

Cohn, Marisa Leavitt, Susan Elliott Sim, and Charlotte P. Lee. 2009. “What Counts as Software Process? Negotiating the Boundary of Software Work Through Artifacts and Conversation.” *Computer Supported Cooperative Work (CSCW)* 18 (5): 401. https://doi.org/10.1007/s10606-009-9100-4.

Collins, Stephanie. 2017. “Care Ethics: The Four Key Claims.” In *Moral Reasoning*, edited by David R. Morrow. New York: Oxford University Press.

“Contribution.” n.d. In *Online Etymology Dictionary*. Douglas Harper. Accessed December 20, 2021. https://www.etymonline.com/word/contribution.

Corbett, Jon. 2021. Interview with Jon Corbett Interview by Daniel Temkin. https://esoteric.codes/blog/jon-corbett.

Couture, Stéphane. 2019. “The Ambiguous Boundaries of Computer Source Code and Some of Its Political Consequences.” In *DigitalSTS: A Field Guide for Science*

& Technology Studies, edited by Janes Vertesi and David Ribes, 136–56. Princeton University Press.

Cowan, Ruth Schwartz. 1985. *More Work For Mother: The Ironies Of Household Technology From The Open Hearth To The Microwave*. 2nd edition. New York, NY: Basic Books.

Dant, Tim. 2010. “The Work of Repair: Gesture, Emotion and Sensual Knowledge.” *Sociological Research Online* 15 (3): 97–118. https://doi.org/10.5153/sro.2158.

Deijnen, Tom van. 2019. In *Festival of Maintenance*. Liverpool.

Deleuze, Gilles. 1988. *Bergsonism*. New York: Zone Books.

Denis, Jérôme, Alessandro Mongili, and David Pontille. 2016. “Maintenance & Repair in Science and Technology Studies.” *TECNOSCIENZA: Italian Journal of Science & Technology Studies* 6 (2): 5–16.

Denis, Jérôme, and David Pontille. 2015. “Material Ordering and the Care of Things.” *Science, Technology, & Human Values* 40 (3): 338–67. https://doi.org/10.1177/0162243914553129.

———. 2019. “Why Do Maintenance and Repair Matter?” In *The Routledge Companion to Actor-Network Theory*, edited by Anders Blok, Ignacio Farías, and Celia Roberts, 238–93. Routledge. https://hal-mines-paristech.archives-ouvertes.fr/hal-02172939v2.

Derrida, Jacques. 1994. *Specters of Marx: The State of the Debt, the Work of Mourning, and the New International*. Translated by Peggy Kamuf. Routledge.

———. 2005. *Paper Machine*. Translated by Rachel Bowlby. Cultural Memory In the Present. Stanford University Press.

Dieter, Michael, Carolin Gerlitz, Anne Helmond, Nathaniel Tkacz, Fernando N. van der Vlist, and Esther Weltevrede. 2019. “Multi-Situated App Studies: Methods and Propositions.” *Social Media + Society* 5 (2): 2056305119846486. https://doi.org/10.1177/2056305119846486.

Dieter, Michael, Carolin Gerlitz, Anne Helmond, Nathaniel Tkacz, Fernando van der Vlist, and Esther Weltevrede. 2018. “Store, Interface, Package, Connection.” *SFB 1187 Medien Der Kooperation - Working Paper Series*, no. 4 (August): 1–16.

Douglass, Jeremy, Mark Marino, and Jessica Pressman. 2020. “Collaborative Reading Praxis.” *Electronic Book Review*, September.

Engelke, Matthew. 2000. “An Interview with Edith Turner.” *Current Anthropology* 41 (5): 843–52. https://doi.org/10.1086/317412.

Ensmenger, Nathan. 2010a. “Making Programming Masculine.” In *Gender Codes: Why Women Are Leaving Computing*, edited by Thomas Misa, 115–41. IEEE Computer Society.

———. 2010b. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. History of Computing. Cambridge, MA, USA: MIT Press.

———. 2014. “When Good Software Goes Bad: The Surprising Durability of an Ephemeral Technology.” In *Mistakes, Ignorance, Contingency and Error in Science and Technology*. Münich.

Ernst, Wolfgang, and Jussi Parikka. 2012. *Digital Memory and the Archive*. Edited by Jussi Parikka. University of Minnesota Press.

Fisher, Mark. 2012. “What Is Hauntology?” *Film Quarterly* 66 (1): 16–24. https://doi.org/10.1525/fq.2012.66.1.16.

Fox, Nick J., and Pam Alldred. 2015. “New Materialist Social Inquiry: Designs, Methods and the Research-Assemblage.” *International Journal of Social Research Methodology* 18 (4): 399–414. https://doi.org/10.1080/13645579.2014.921458.

Gad, Christopher, and Casper Bruun Jensen. 2016. “Lateral Concepts.” *Engaging Science, Technology, and Society* 2 (May): 3–12. https://doi.org/10.17351/ests2016.77.

Galloway, Alexander R. 2004. *Protocol: How Control Exists after Decentralization*. Leonardo. Cambridge, MA, USA: MIT Press.

Garfinkel, Harold. 1967. *Studies in Ethnomethodology*. Englewood Cliffs, N.J.: Prentice-Hall.

———. 1996. “Ethnomethodology’s Program.” *Social Psychology Quarterly* 59 (1): 5–21. https://doi.org/10.2307/2787116.

Geertz, Clifford. 1973. “Thick Description: Towards an Interpretive Theory of Culture.” In *The Interpretation of Cultures*. Basic Books.

Geiger, R. Stuart, Dorothy Howard, and Lilly Irani. 2021. “The Labor of Maintaining and Scaling Free and Open-Source Software Projects.” *Proceedings of the ACM on Human-Computer Interaction* 5 (CSCW1): 1–28. https://doi.org/10.1145/3449249.

Giddens, Anthony. 1984. *The Constitution of Society: Outline of the Theory of Structuration*. Polity.

Gielen, P.J.D., and P. de Bruyne. 2012. “Introduction. The Catering Regime.” In *Teaching Art in the Neoliberal Realm. Realism versus Cynicism*, edited by Pascal Gielen and Paul De Bruyne, 1–11. Amsterdam: Valiz.

Glaser, Barney G, and Anselm L Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine.

Godfrey-Smith, Peter. 2003. *Theory and Reality: An Introduction to the Philosophy of Science*. Science and Its Conceptual Foundations Series. Chicago, IL: University of Chicago Press.

Gorm, Nanna, and Irina Shklovski. 2019. “Episodic Use: Practices of Care in Self-Tracking.” *New Media & Society* 21 (11–12): 2505–21. https://doi.org/10.1177/1461444819851239.

Graham, Stephen, and Nigel Thrift. 2007. “Out of Order: Understanding Repair and Maintenance.” *Theory, Culture & Society* 24 (3): 1–25. https://doi.org/10.1177/0263276407075954.

Haigh, T. 2009. “How Data Got Its Base: Information Storage Software in the 1950s and 1960s.” *IEEE Annals of the History of Computing*. https://doi.org/10.1109/MAHC.2009.123.

Hallé, Clémence, and Anne-Sophie Milon. 2020. “The Infinity of the Anthropocene: A (Hi)Story with a Thousand Names.” In *Critical Zones. The Science and Politics of Landing on Earth*, edited by Bruno Latour and Bruno Weibel, 560. MIT Press.

Hammersley, Martyn, and Paul Atkinson. 2007. *Ethnography: Principles in Practice, 3rd Edition*. 3rd edition. London ; New York: Routledge.

Haraway, Donna. 1985. “A Manifesto for Cyborgs: Science, Technology, and Socialist Feminism in the 1980s.” *Socialist Review*, no. 80: 7–44. https://doi.org/10.1080/08164649.1987.9961538.

———. 1988. “Situated Knowledges: The Science Question in Feminism and the Privilege of Partial Perspective.” *Feminist Studies* 14 (3): 575–99. https://doi.org/10.2307/3178066.

———. 1997. *Modest_Witness@Second_Millennium.Femaleman_Meets_Oncomouse: Feminism and Technoscience*. Routledge.

———. 2016. *Staying with the Trouble. Making Kin in the Chthulucene*. Experimental Futures. https://www.dukeupress.edu/staying-with-the-trouble.
Heidegger, Martin. 1977. *Question Concerning Technology, and Other Essays, The*. 12.2.1976 edition. New York, NY: Harper Torchbooks.

———. 1996. *Being and Time: A Translation of Sein Und Zeit*. SUNY Press.

Hendricks, Vincent, and Pelle Hansen. 2016. *Infostorms. Why Do We “Like”? Explaining Individual Behaviour on the Social Net*. Copernicus. Springer.

Henke, Christopher R. 1999. “The Mechanics of Workplace Order: Toward a Sociology of Repair.” *Berkeley Journal of Sociology* 44: 55–81.

Hicks, Mar. 2017. *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*. History of Computing. Cambridge, MA, USA: MIT Press.

Horkheimer, Max. 1972. “Traditional and Critical Theory.” In *Critical Theory. Selected Essays*, translated by Matthew O’Connell, 188–243. The Continuum Publishing Company.

Høyer Leivestad, Hege, Anette Nyqvist, and Hans Tunestad. 2017. “Individuals and Industries : Large-Scale Professional Gatherings as Ethnographic Fields.” In *Ethnographies of Conferences and Trade Fairs: Shaping Industries, Creating Professionals*, edited by Hege Høyer Leivestad and Anette Nyqvist, 1–21. Palgrave Macmillan.

Hui, Yuk. 2016. *On the Existence of Digital Objects*. University of Minnesota Press.

Hunt, Andrew, and David Thomas. 1999. *The Pragmatic Programmer: From Journeyman to Master*. 1st edition. Reading, Mass: Addison-Wesley Professional.

Ihde, Don. 1993. *Postphenomenology: Essays in the Postmodern Context*. Northwestern University Press.

Ingold, Tim. 2000. *The Perception of the Environment: Essays on Livelihood, Dwelling and Skill*. 1st edition. London ; New York: Routledge.

———. 2007. “Materials against Materiality.” *Archaeological Dialogues* 14 (1): 1–16. https://doi.org/10.1017/S1380203807002127.

———. 2010. “Bringing Things Back to Life: Creative Entanglements in a World of Materials.” Working Paper. Realities / Morgan Centre, University of Manchester.

Ingwersen, Peter, and Kalervo Järvelin. 2005. *The Turn. Integration of Information Seeking and Retrieval in Context*. Vol. 18. The Information Retrieval Series. Springer.

Jackson, Steven J. 2014. “Rethinking Repair.” In *Media Technologies: Essays on Communication, Materiality, and Society*. The MIT Press. https://doi.org/10.7551/mitpress/9780262525374.003.0011.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2017. *An Introduction to Statistical Learning: With Applications in R*. Corr. 7th printing. New York: Springer.

Jansen, K., and S. Vellema. 2011. “What Is Technography?” *NJAS - Wageningen Journal of Life Sciences*, Technography and Interdisciplinarity: Performance, Practices and Experiments, 57 (3): 169–77. https://doi.org/10.1016/j.njas.2010.11.003.

Johnson, Lee. 2016. “All Software Is Legacy.” February 22, 2016. https://leejo.github.io/2016/02/22/all_software_is_legacy/.

Kaldrak, Irina, and Martina Leeker. 2015. “There Is No Software, There Are Just Services: Introduction.” In *There Is No Software, There Are Just Services*, 9–20. Digital Cultures. Lüneburg: meson press.

Kelty, Christopher. 2008. *Two Bits: The Cultural Significance of Free Software*. Duke University Press.

Kelty, Christopher, and Seth Erickson. 2015. “Durability of Software.” In *There Is No Software, There Are Just Services*, edited by Irina Kaldrak and Martina Leeker, 39–56. Digital Cultures. Lüneburg: meson press.

Kirschenbaum, Matthew. 2016. *Track Changes. A Literary History of Word Processing*. Harvard University Press.

Kittler, Friedrich. 1992. *Discourse Networks, 1800/1900*. Translated by Michael Metteer. 1st edition. Stanford, Calif: Stanford University Press.

———. 1995. “There Is No Software.” *CTHEORY*, October. http://www.ctheory.net/articles.aspx?id=74.

———. 1999. *Gramophone, Film, Typewriter*. Translated by Geoffrey Winthrop-Young and Michael Wutz. Stanford: Stanford University Press.

Kjær, Katrine Meldgaard, Mace Ojala, and Line Henriksen. 2021. “Absent Data: Engagements with Absence in a Twitter Collection Process.” *Catalyst: Feminism, Theory, Technoscience* 7 (2). https://doi.org/10.28968/cftt.v7i2.34563.

Knauft, Bruce M. 2006. “Anthropology in the Middle.” *Anthropological Theory* 6 (4): 407–30. https://doi.org/10.1177/1463499606071594.

Knuth, David. 1984. “Literate Programming.” *The Computer Journal* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Kocksch, Laura, Matthias Korn, Andreas Poller, and Susann Wagenknecht. 2018. “Caring for IT Security: Accountabilities, Moralities, and Oscillations in IT Security Practices.” *Proceedings of the ACM on Human-Computer Interaction* 2 (CSCW): 92:1-92:20. https://doi.org/10.1145/3274361.

Krämer, Sybille. 2006. “The Cultural Techniques of Time Axis Manipulation: On Friedrich Kittler’s Conception of Media.” *Theory, Culture & Society* 23 (7–8): 93–109. https://doi.org/10.1177/0263276406069885.

Lacan, Jacques. 2006. “The Direction of the Treatment and the Principles of Its Power.” In *Écrits: The First Complete Edition in English*, translated by Bruce Fink, 489–542. W. W. Norton.

Laet, Marianne de, and Annemarie Mol. 2000. “The Zimbabwe Bush Pump: Mechanics of a Fluid Technology.” *Social Studies of Science* 30 (2): 225–63. https://doi.org/10.1177/030631200030002002.

Landow, George. 2006. *Hypertext 3.0. Critical Theory and New Media in an Era of Globalization*. Johns Hopkins University Press Books.

Latour, Bruno. 1987. *Science in Action: How to Follow Scientists and Engineers*. Harvard University Press.

———. 1990. “Technology Is Society Made Durable.” *The Sociological Review* 38 (1_suppl): 103–31. https://doi.org/10.1111/j.1467-954X.1990.tb03350.x.

———. 2007. *Reassembling the Social: An Introduction to Actor-Network-Theory*. Clarendon Lectures in Management Studies. Oxford, New York: Oxford University Press.

———. 2012. *An Inquiry into Modes of Existence*. Harvard University Press.

Latour, Bruno, and Steve Woolgar. 1979. *Laboratory Life. The Construction of Scientific Facts*. SAGE Publications.

Law, John. 2004a. *After Method: Mess in Social Science Research*

———. 2004b. *After Method: Mess in Social Science Research*. 1st edition. London ; New York: Routledge.

———. 2010. “Care and Killing: Tensions in Veterinary Practice.” In , edited by Annemarie Mol, Ingunn Moser, and Jeannette Pols, 57–69. Bielefeld: Transcript.

Law, John, Evelyn Ruppert, and Mike Savage. 2011. “The Double Social Life of Methods.” Working Paper 95. CRESC Working Paper Series. Open University.

Lee, Francis, and Lotta Björklund Larsen. 2019. “How Should We Theorize Algorithms? Five Ideal Types in Analyzing Algorithmic Normativities.” *Big Data & Society* 6 (2): 2053951719867349. https://doi.org/10.1177/2053951719867349.

Lessig, Lawrence. 1999. *Code: And Other Laws Of Cyberspace*. First Edition. New York: Basic Books.

Lievrouw, Leah A. 2014. “Materiality and Media in Communication and Technology Studies: An Unfinished Project.” In *Media Technologies*, edited by Tarleton Gillespie, Pablo Boczkowski, and Kirsten A. Foot. The MIT Press.

Light, Ben, Jean Burgess, and Stefanie Duguay. 2018. “The Walkthrough Method: An Approach to the Study of Apps.” *New Media & Society* 20 (3): 881–900. https://doi.org/10.1177/1461444816675438.

Lindén, Lisa, and Doris Lydahl. 2021. “Editorial: Care in STS.” *Nordic Journal of Science and Technology Studies*, April, 3–12. https://doi.org/10.5324/njsts.v9i1.4000.

Locke, John. 1689. *Two Treatises of Government: In the Former, The False Principles, and Foundation of Sir Robert Filmer, and His Followers, Are Detected and Overthrown. The Latter Is an Essay Concerning The True Original, Extent, and End of Civil Government*. Awnsham Churchill.

Lupton, Deborah. 2020. *The Sociology of Mobile Apps*.

Luu, Trieuvy, Martijn van den Broeck, and Marie Louise Juul Søndergaard. 2018. “Data Economy: Interweaving Storytelling and World Building in Design Fiction.” In *Proceedings of the 10th Nordic Conference on Human-Computer Interaction - NordiCHI ’18*, 771–86. Oslo, Norway: ACM Press. https://doi.org/10.1145/3240167.3240270.

Lynch, Gawain, Erin Taylor, and Don Goodman-Wilson. 2020. “Maintainerati Berlin 2019 Event Report.” The Hague: Maintainerati Foundation. https://maintainerati.org/blog/berlin-2019-event-report/.

Mackenzie, Adrian. 2006. *Cutting Code. Software and Sociality*. Vol. 30. Digital Formations. Peter Lang.

———. 2013. “Programming Subjects in the Regime of Anticipation: Software Studies and Subjectivity.” *Subjectivity* 6 (4): 391–405. https://doi.org/10.1057/sub.2013.12.

———. 2017. *Machine Learners*. MIT Press.

Mahoney, Michael S. 2008. “What Makes the History of Software Hard.” *IEEE Annals of the History of Computing* 1 (3): 8–18.
“Maintainerati Berlin 2019 Session Notes.” 2020. The Hague: Maintainerati Foundation. https://maintainerati.org/blog/berlin-2019-event-report/.

Manovich, Lev. 2013. *Software Takes Command*. International Texts in Critical Media Aesthetics. Bloomsbury Academic.

———. 2015. “Data Science and Digital Art History.” *International Journal for Digital Art History*, no. 1 (June). https://doi.org/10.11588/dah.2015.1.21631.

Marcus, George. 1995. “Ethnography in/of the World System: The Emergence of Multi-Sited Ethnography.” *Annual Review of Anthropology* 24: 95–117.

Marino, Mark. 2012. “Critical Code Studies.” *Electronic Book Review*, January. https://electronicbookreview.com/essay/critical-code-studies/.

———. 2020. *Critical Code Studies*. MIT Press.

Mattern, Shannon. 2018. “Maintenance and Care.” *Places Journal*, November. https://doi.org/10.22269/181120.

McLuhan, Marshall. 1962. *The Gutenberg Galaxy: The Making of Typographic Man*. Toronto: Universtity of Toronto Press.

Mol, Annemarie. 2003. *The Body Multiple. Ontology in Medical Practice*. Duke University Press.

———. 2008. *The Logic of Care: Health and the Problem of Patient Choice*. Routledge..

Mol, Annemarie, Ingunn Moser, and Jeannette Pols. 2010. “Care in Practice: On Tinkering in Clinics, Homes and Farms.” *Care in Practice* 8 (December). https://doi.org/10.14361/transcript.9783839414477.

Neuman, Lawrence. 2007. *Basics of Social Research: Qualitative and Quantitative Approaches*. 2nd International Edition. Pearson.

Newton, Isaac. 1687. *Philosophiæ Naturalis Principia Mathematica*.

Noble, Safiya Umoja. 2018. *Algorithms of Oppression*. New York University Press.

Nova, Nicolas, and Anaïs Bloch. 2020. *Dr. Smartphones: An Ethnography of Mobile Phone Repair Shops*. IDP. https://hal.archives-ouvertes.fr/hal-03106034.

Nyce, James M., and Sanna Talja. 2015. "The Problem with Problematic Situations: Differences between Practices, Tasks and Situations as Units of Analysis." *Library & Information Science Research* 37 (1): 61–67.

Ojala, Mace. 2020. "Experiencing Collective Accounts of 'Touch': Analyzing Software Maintainers Just Speak." In . RUSTlab Lectures. Bochum: Ruhr Universität Bochum.

O'Neil, Cathy. 2016. *Weapons of Math Destruction: How Big Data Increases Inequality and Threatens Democracy*. 1st edition. New York: Crown.

Orr, Julian E. 1996. *Talking about Machines: An Ethnography of a Modern Job*. 1st edition. Ithaca, N.Y: ILR Press.

———. 2006. "Ten Years of Talking About Machines." *Organization Studies* 27 (12): 1805–20. https://doi.org/10.1177/0170840606071933.

Parikka, Jussi. 2012. *What Is Media Archaeology?* 1st edition. Cambridge, UK ; Malden, MA: Polity.

Pavese, Carlotta. 2021. "Knowledge How." In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Summer 2021. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2021/entries/knowledge-how/.

Peirce, C.S. 1935. "How to Make Our Ideas Clear." In *Collected Papers of Charles Sanders Peirce, Volumes V and VI: Pragmatism and Pragmaticism and Scientific Metaphysics*, edited by Charles Hartshorne and Paul Weiss. Harvard University Press.

Peters, Tim. 2004. "PEP 20. The Zen of Python." Python Software Foundation. https://www.python.org/dev/peps/pep-0020/.

Piccini, Angela. 2015. "Media-Archaeologies: An Invitation." *Journal of Contemporary Archaeology*, Forum, 2 (1): 1–7.

Pinch, Trevor J., and Wiebe E. Bijker. 1984. "The Social Construction of Facts and Artefacts: Or How the Sociology of Science and the Sociology of Technology Might Benefit Each Other." *Social Studies of Science* 14 (3): 399–441. https://doi.org/10.1177/030631284014003004.

Pink, Sarah, Minna Ruckenstein, Robert Willim, and Melisa Duque. 2018. “Broken Data: Conceptualising Data in an Emerging World.” *Big Data & Society* 5 (1): 2053951717753228. https://doi.org/10.1177/2053951717753228.

Plant, Sadie. 1997. *Zeros + Ones: Digital Women and the New Technoculture*. London: Fourth Estate.

Polanyi, Michael. 1958. *Personal Knowledge: Towards a Post-Critical Philosophy*. Chicago: University of Chicago Press.

Porter, Theodore. 1995. *Trust in Numbers: The Pursuit of Objectivity in Science and Public Life*. Princeton University Press.

Puig de la Bellacasa, Maria. 2011. “Matters of Care in Technoscience: Assembling Neglected Things.” *Social Studies of Science* 41 (1): 85–106. https://doi.org/10.1177/0306312710380301.

———. 2017. *Matters of Care. Speculative Ethics in More Than Human Worlds*. University of Minnesota Press.

Rebaudengo, S. 2012a. “Addicted Product: A Scenario of Future Interactions Based on Addictions of Products to Be Used.” Master Thesis, Delft: TU Delft. https://repository.tudelft.nl/islandora/object/uuid%3A5d08ae4d-c551-458a-9ddd-d2b9f9e8e233.

———. 2012b. “Addicted Products.” 2012. http://www.simonerebaudengo.com/project/addictedproducts.

Reckwitz, Andreas. 2002. “Toward a Theory of Social Practices: A Development in Culturalist Theorizing.” *European Journal of Social Theory* 5 (2): 243–63. https://doi.org/10.1177/13684310222225432.

Reeves-Evison, Theo, and Mark Justin Rainey. 2018. “Ethico-Aesthetic Repairs.” *Third Text* 32 (1): 1–15. https://doi.org/10.1080/09528822.2018.1448516.

Rieder, Bernhard. 2020. *Engines of Order. A Mechanology of Algorithmic Techniques*. Amsterdam University Press.

Ritasdatter, Linda Hilfling. 2020. “Unwrapping Cobol: Lessons in Crisis Computing.” PhD Thesis, Malmö: Malmö University. https://doi.org/10.24834/isbn.9789178771165.

Ryding, Karin. 2020. “The Silent Conversation: Designing for Introspection and Social Play in Art Museums.” In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–10. CHI ’20. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3313831.3376357.

Saariketo, Minna, and Mareike Glöss. 2020. “In the Grey Zone of Hacking? Two Cases in the Political Economy of Software and the Right to Repair.” In . Hacker Cultures. Understanding the Actors Behind Our Technology. Prague: European Association for the Study of Science and Technology. https://www.buzzsprout.com/1323889/5257063.

Schwartzman, Helen. 1989. *The Meeting: Gatherings in Organizations and Communities*. Springer.

Sicart, Miguel, and Irina Shklovski. 2020. “’Pataphysical Software: (Ridiculous) Technological Solutions for Imaginary Problems.” In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, 1859–71. DIS ’20. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3357236.3395526.

Sims, Benjamin, and Christopher R. Henke. 2012. “Repairing Credibility: Repositioning Nuclear Weapons Knowledge after the Cold War.” *Social Studies of Science* 42 (3): 324–47. https://doi.org/10.1177/0306312712437778.

Sipser, Michael. 2012. *Introduction to the Theory of Computation*. 3rd edition. Boston, MA: Cengage Learning.

Star, Susan Leigh. 1999. “The Ethnography of Infrastructure.” *American Behavioral Scientist* 43 (3): 377–91. https://doi.org/10.1177/00027649921955326.

Star, Susan Leigh, and Karen Ruhleder. 1996. “Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces.” *Information Systems Research* 7 (1): 111–34. https://doi.org/10.1287/isre.7.1.111.

Sterne, Jonathan. 2012. *MP3. The Meaning of a Format*. Sign, Storage, Transmission. Duke University Press.

Stevenson, Michael, and Anne Helmond. 2020. “The Historical Trajectories of Algorithmic Techniques: An Interview with Bernhard Rieder.” *Internet Histories* 4 (1): 105–14. https://doi.org/10.1080/24701475.2020.1723345.

Suchman, Lucy. 2002. “Located Accountabilities in Technology Production.” *Scandinavian Journal of Information Systems* 14 (2). https://aisel.aisnet.org/sjis/vol14/iss2/7.

———. 2012a. “Configuration.” In *Inventive Methods*. Routledge.

———. 2012b. *Human-Machine Reconfigurations: Plans and Situated Actions*. 2nd ed. Cambridge University Press.

Terpsma, Florian Cramer, and Nienke Terpsma. 2021. “What Is Wrong with the Vienna Declaration on Artistic Research?” *Open!* (blog). 2021. https://onlineopen.org/what-is-wrong-with-the-vienna-declaration-on-artistic-research.

Traweek, Sharon. 1992. *Beamtimes and Lifetimes. The World of High Energy Physicists*. Harvard University Press.

Tronto, Joan. 1993. *Moral Boundaries: A Political Argument for an Ethic of Care*. Routledge.

Tsing, Anna Lowenhaupt. 2015. *The Mushroom at the End of the World: On the Possibility of Life in Capitalist Ruins*.

Tuin, Iris van der. 2014. “Diffraction as a Methodology for Feminist Onto-Epistemology: On Encountering Chantal Chawaf and Posthuman Interpellation.” *Parallax* 20 (3): 231–44. https://doi.org/10.1080/13534645.2014.927631.

Turing, A. M. 1937. “On Computable Numbers, with an Application to the Entscheidungsproblem.” *Proceedings of the London Mathematical Society* s2-42 (1): 230–65. https://doi.org/10.1112/plms/s2-42.1.230.

Tuschling, Anna. 2016. “Historical, Technological and Medial a Priori: On the Belatedness of Media.” *Cultural Studies* 30 (4): 680–703. https://doi.org/10.1080/09502386.2016.1180759.

Vanderschraaf, Peter, and Giacomo Sillari. 2021. “Common Knowledge.” In *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta, Fall 2021. Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/fall2021/entries/common-knowledge/.

Verbeek, Peter P. C. C. 2001. “Don Ihde: The Technological Lifeworld.” In *American Philosophy of Technology: The Empirical Turn*, edited by Huis Achterhuis, 119–46. Indiana University Press.

“Vienna Declaration on Artistic Research.” 2020. AEC, CILECT / GEECT, Culture Action Europe, Cumulus, EAAE, ELIA, EPARM, EQ-Arts, MusiQuE and SAR. https://cultureactioneurope.org/news/vienna-declaration-on-artistic-research/.

Vinsel, Lee, and Andrew Russell. 2020. *The Innovation Delusion. How Our Obsession with the New Has Disrupted the Work That Matters Most*. Currency.

Wenger, Etienne. 1998. *Communities of Practice: Learning, Meaning, and Identity*. Communities of Practice: Learning, Meaning, and Identity. New York, NY, US: Cambridge University Press. https://doi.org/10.1017/CBO9780511803932.

Wilde, Mandy de. 2021. "'A Heat Pump Needs a Bit of Care': On Maintainability and Repairing Gender–Technology Relations." *Science, Technology, & Human Values* 46 (6): 1261–85. https://doi.org/10.1177/0162243920978301.

Winner, Langdon. 1980. "Do Artifacts Have Politics?" *Daedalus* 109 (1): 121–36.

Winthereik, Brit Ross. 2019. *Is ANT's Radical Empiricism Ethnographic?* Routledge Handbooks Online. https://doi.org/10.4324/9781315111667-4.

Wright, Alex. 2014. *Cataloging the World: Paul Otlet and the Birth of the Information Age*. Oxford, New York: Oxford University Press.

Zhou, Shurui, Bogdan Vasilescu, and Christian Kästner. 2020. "How Has Forking Changed in the Last 20 Years? A Study of Hard Forks on GitHub." In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 445–56. ICSE '20. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3377811.3380412.

Ziewitz, Malte. 2016. "Governing Algorithms: Myth, Mess, and Methods." *Science, Technology, & Human Values* 41 (1): 3–16. https://doi.org/10.1177/0162243915608948.

Zuboff, Shoshana. 2019. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. 1st edition. New York: PublicAffairs.