

Reading Code Aloud

Mace Ojala <mace_dot_ojala_at_rub_dot_de>, Ruhr University Bochum  <https://orcid.org/0000-0002-9546-1736>

DOI: <https://doi.org/10.63744/ae5a4saj2fhh>

Abstract

In this article, I present reading computer source code aloud, and reflect on it together in dialogue, as a method which offers new forms of engagement with code and programming. With my peer discussion partner, we experimented with this methodology by exchanging recorded audio messages in a research project involving computational data-analysis. We discovered that abandoning the screen allowed us to de-explain, de-familiarize and de-contextualize code we had programmed, and negotiate new, shared insight of it as a cultural and situated object through re-explanation, re-familiarization, and re-contextualization of our experience of code.

Prologue: reading code aloud

"But this, like, changing to the audio realm, is doing some really strange things for me."

In this article, I present reading computer source code aloud in dialogue as a method which offers new, reflexive forms of engagement with code and programming. With my good colleague Katrine Meldgaard Kjær, we experimented with this methodology as a part of a research project which involved code as a form of analysis, and a collaboration between a researcher with programming expertise and a qualitatively trained researcher. As a response to on one hand feeling constrained by our lack of a common language to understand code, and to the conditions of collaboration during COVID-19 social distancing on the other, we used an exchange of recorded audio messages as a way to elicit reflection on the data-analysis code we had programmed in our project. We explicitly designed the method to background the instrumental use of code, and foreground its wider connotative meanings to us, and the associations and reflections reading and hearing code provoked in us. In the following, we argue that moving away from the screen into audio and dialogue offers an arena for new types of engagement with code, and contributes to development of "a library of methods in an ever-expanding set of methods" in the emergent field of critical code studies [Marino 2020]. In particular I propose that reading code aloud together offers a possibility of developing new understanding of code as a collective and communal practice among collaborating partners, as well as a way of highlighting the situated and contextual nature of code and programming.

The method was relatively simple: using a smartphone, I recorded myself reading aloud a central piece of code from the research project, and sent it to my colleague Katrine as an email attachment. She responded with her reflections upon hearing the code being read aloud, replying with a voice message. I recorded myself then replied with reflections in turn prompted in them. This exchange of voice messages was repeated back-and-forth, initially for a couple of weeks. In this way, what took form was an extended, slowed-down conversation about the ways in which reading this code and hearing it being read aloud affected us, and how we related to this specific piece of code, and to each other's

reflections.

I attach below the code segment from our research project as the basis of our methodological suggestions and explorations throughout the article. The research project sought to analyse the development of language used in relation to the topic of medicinal cannabis on the Danish internet. For this purpose, we collected a corpus of around 8 million topic-relevant documents from the Danish internet archive *Netarkivet*, with the assistance of the Danish Royal Library. The project was inspired by corpus linguistics and computational analysis of big textual data [Baker 2012]; [Baker et al. 2008]. Data-analysis was done on a computation platform the data-owner Danish Royal Library provided. The platform consisted of virtual server on a private cloud, serving the popular RStudio ([RStudio Team 2020]) interface over the web to the R programming language [R Core Team 2020]. The corpus was stored on HDFS storage, and made accessible to us in a Spark computation engine [Zaharia et al. 2026] which I interfaced with the infrastructure from the virtual RStudio by bricolaging literary programming [Knuth 1984] using R Markdown notebooks [Xie et al. 2018] and interactive run-evaluate-print-loop (REPL) style programming. We return to affective implications of this infrastructure below (Example 1).

3

```
01     cannabis_mc %>%  
02       select(crawl_year, domain) %>%  
03       group_by(crawl_year, domain) %>%  
04       tally %>%  
05       collect %>%  
06       group_by(crawl_year) %>%  
07       mutate(rank = rank(desc(n), ties.method = 'first')) %>%  
08       arrange(crawl_year, rank) %>%  
09       select(rank, crawl_year, domain) %>%  
10       pivot_wider(names_from = crawl_year, values_from = domain) %>%  
11       write_csv2('../domains-medicinsk_cannabis-freq_by_annum.csv')
```

Throughout the remainder of the article, I will present, contextualize and share our reflections of the code in Example 1 through our use of the method of reading code aloud. I will also present and discuss excerpts from computationally produced transcripts of our voice messages to each other; these will be interlaced with our discussion of the method more generally throughout the article. The text below is not presented in the traditional format of social science articles, and this is intentional. I, nor we, do not neatly separate theory, methods, and analysis sections in the below, but rather present reading code aloud in terms of onto-epistem-ology, that is the study of practices of knowledges in being [Barad 2007, 185]. As such, theory, method and analysis are entangled throughout the article, which is structured in three main movements. These movements represent key elements in what we believe reading and active listening allows for: it de- and re-explains; it de- and re-familiarizes; and it de- and re-contextualized code. Through these movements, I present how reading code aloud centres an affective and situated analysis of code itself rather than an instrumental analysis of code's functionality. This is explored via a reflexive account and considerations of, among other themes, (personal) histories of code and coding, how material artifacts involved in separating 'code' from 'embodiment', and labour divisions in interdisciplinary research projects. Proposing reading code aloud, we interlace our text with dialogue and source code, and thereby draw attention back to analysis programming as a reflexive practice.

4

Introduction: interdisciplinary struggles in Digital Humanities

In our research project, we sought to use computational methods to open insights into the discursive developments in the digital life of medicinal cannabis debates. For this purpose, we teamed up across complementary areas of expertise: Katrine had knowledge about the medicinal cannabis case, and I had knowledge of how to use code to computationally analyse the data, but no issue-specific expertise in relation to medicinal cannabis. While we share a lot of common academic, institutional, practical and social ground through having read some of the same academic literature, employment in the same science and technology studies (STS) oriented research group Technologies in Practice and its associated research lab ETHOSlab, and attending the same awkward Christmas parties at work, it quickly became

5

clear that our different positions were not always conceptually aligned, and that there was a knowledge gap between us; Katrine sometimes lacked insight into what the specific pieces of code signified, and because I did not have deep knowledge of the case, I frequently lacked insight into the reasoning behind my colleague's analytical aims and interpretations as I was writing code.

This division of labour aligns with common patterns in this type of collaborative work, which is also a persisting theme in Digital Humanities: scholars here have argued that much research between quantitative or computational and qualitative researchers consists of monodisciplinary collaboration between several parties, where humanities and social sciences are often seen as a data source, while scholars adept in computational methods are as seen as tool providers [Bradley et al. 2018];[Terras et al. 2013]. Within these collaborations, however, there is limited potential for mutual transformation and development of knowledge traditions for both parties, as each stay in their lane of expertise (ibid.). Indeed, while there is much call for interdisciplinary work to engage with “grand challenges”, suggestions for methods for how to work together across disciplinary divides are still largely missing or vague [Bornakke and Due 2018]. This implies that much of the work to foster and sustain interdisciplinary collaboration becomes invisible or unarticulated [Graban et al. 2019].

The division of labour and expertise in our project team also reflects the contemporary gendered dimensions of programming work. Together we, Mace and Katrine, typify the gendered expectations and coding of contemporary programming work. Mace, who has the technical coding expertise, identifies as a man; and Katrine, who has issue expertise, as a woman. This reflects the cultural status of programming as masculine dominated, and some have argued, a largely “bro”-ified domain [Hicks 2013]. The historical gendering of computing throughout the second half the 20th century is by now well documented, as well as how women have been foundational, yet also erased from computing history [Hicks 2017]. Gender and programming are thus fundamentally connected to issues of power, in particular in relation to the development of software engineering as a distinct technoscientific discipline [Abbate 2012]; [Abbate 2021];[Ensmenger 20010];[Hicks 2013]. Programming has been isolated from various communities and “domains” computing it is applied in and it draws from various academic disciplines such as geo, bio and other Sciences, artificial intelligence, linguistics as well as business data processing, management and warfare [Mahoney 2005];[Ribes 2019];[Ribes et al. 2019];[Terras et al. 2013].

In our research project, we wanted to counter our disciplinary and gendered divergence, and to both develop our respective standings in our knowledge traditions in order to foster better collaboration. To do so, however, it became clear that we needed to work with code in a different way than our usual, screen-centric practices of simply sitting together to discuss the written code. As the issue expert Katrine found the discussion about the project exclusively anchored in the many lines of code on screen to be both overwhelming and limiting, while for myself the code was so entangled with the data, the infrastructures at hand, and cultures of computation that discussing it without this wider context seemed more confusing than helpful. Early on, these reactions pointed to the capacity of code to cause emotional reactions; code here was not a “dead”, neutral or ready-to-hand tool, but something that was in the process of becoming, and as such affected us both, albeit in different ways. This realization inspired us to change perspectives from being interested in analysing data with code, to analysing code as data itself. After working with design and creative methods such as sketching on pen and paper ([Buxton 2007];[Kirk 2019]) and while the social distancing measures of the COVID-19 pandemic were prolonged, we decided to turn away from the written medium altogether, and instead use audio as the medium of understanding code. In turning to reading and reflecting on code aloud, away from the screen, we found that the resonance of the sonic ([Nancy 2007]) enabled three key and interconnected openings: a departure from instrumental use and interpretation of code, the development of a new common language for speaking about code, and a situated approach to understanding the effects and affects of code. These, in turn, were related to processes of de- and re-explanation, de- and re-familiarizing, and de- and re-contextualizing of code. Based on our discussions, in the following I unpack each of these three aspects of reading code aloud as method for a critical code studies analysis.

De/re-explaining: reading aloud as a turning away from instrumental and toward cultural text and social context

“Incidentally, when you mentioned that you’ve never heard code being read aloud (...) I was quite surprised to hear that because I remember it as though you said that this was an established method, which is why I thought, initially that, Oh, this is something we could do.”

Reading code aloud offers the opportunity to turn away from established approaches for instrumental explanations of code and instead towards interpreting code from a cultural and social point of departure. When proposing reading code aloud as a method for critical code studies, we are drawing inspiration from established methods from other research fields, in particular from arts-based methods, cultural studies and sound studies. By mobilizing these perspectives in a new context, we are responding to critical code studies’ call for work that remedies “a lack of critical methodologies and vocabulary for exploring code through this cultural lens.” [Marino 2014]. I and we apply these to traditions within programming practice, as well as scholarship on programming and software, in order to open towards a sensitivity for connotations and multiplicity of interpretations. This runs counter to the tradition of seeking clarity and singular meaning in code. In programming praxis, vocalization techniques such as pair-programming ([Mackenzie 2006, 153–158]), quite concretely explaining code to a mute rubber duck ([Hunt and Thomas 1999]) and reading code aloud for programming education ([Swidan 2019]) are used to improve code clarity, sort out thoughts and improve comprehension. In a different vein, in recent years we have seen a push for demanding explanations of code and algorithms to non-programmers [O’Neil 2016];[Diakopoulos 2015];[Kearns and Roth 2019]. While the formerly mentioned vocalization techniques in “extreme programming” ([Mackenzie 2006]) and education can improve code quality and comprehension, and the latter is an important project for fairness, accountability and transparency in the name of social justice, neither does very much to open interpretations of code; indeed, quite the opposite, as both seek a single truth and privilege establishing or recovering the function of the code. The former are utilitarian techniques to negotiate a convergence toward a preferred function to express and write into new code. The latter seeks to make available and contestable the social and political function of existing code. However, a cyclopic and suspicious focus on a small selection of megacorps arguably contributes to their fetishization and the *algorithmic drama* ([Ziewitz 2016]), in turn masking much of the real computational culture outside these megacorps. In particular it misses engagement with code as a cultural object, with cultural functions, and how programming is a practice embedded in specific social contexts.

In order to be able to analyze this relationship between code, culture and the social, there is thus a need for approaches that analyze software both as a text and as a part of a cultural and social context – without reducing it to its mechanics. Critical code studies work is characterized by studying code through a cultural lens with hermeneutic methodologies from literature and textual analysis [Marino 2020]. Informed by our shared work, I would like to contribute to this programme by developing new methodologies for including the social embeddedness of code in ways which bridge the author/audience (or author/analyst) gap, allow the code authors too to participate in interpretation, and recognize that computer code is written not only by dedicated “programmers”, but by diverse actors such as hobbyists, children, actual and aspiring artists, scientists, criminals, military personnel, and media theorists ([Marino 2020]) – and by STS scholars (see Example 1). With this aim, we draw from fields that have already developed methods for opening up to multiple interpretations. In particular, we propose drawing firstly on cultural studies’ use of textual analysis as a tool to examine underlying cultural assumptions of a text and the cultural context in which it is situated ([Fürsich 2009]), secondly from the creative methods that mark in particular critical and feminist arts-based research, which focus on developing new methods as a way of enabling an analysis of what often remains invisible in conventional research practices, in particular the role of the researcher themselves. Moreover, we draw inspiration from feminist STS’ and critical data studies’ focus on the situated nature of knowledge production and critique of assumptions of code and data as disembodied or neutral entities. As a method, reading code aloud is thus a product of the combination of these three fields’ sensitivities toward the text, toward the context of this text, and towards transmedial, phenomenological and intersubjective effects of reading aloud and listening.

From arts-based methods, we draw on the argument that the aim to develop a new critical vocabulary requires developing new and creative methods and approaches. Arts-based methods, not to be confused with artistic research, is an umbrella term for research that uses “artistic forms and expression to explore, understand, represent, and even

challenge human experiences" [Wang et al. 2017]. Arts-based methods have in large part been developed as a response to a so-called "crisis of representation" in anthropology and related qualitative fields, revolving around a critique of the lack of capacity of methods in these fields to capture lived experience in their interpretive accounts ([Schwandt 2007]), as well as the invisibility of the researcher themselves in said accounts. In response to this crisis, particularly ethnographic research has developed new methods of social analysis by mobilizing creative traditions such as storytelling, collaging or poetry in their research (as in eg. [Ceglowski 1997];[Elbow 1997];[Ellis 2004]). Regarding sound, Krogh-Groth and Samson [Krogh Groth and Samson 2016] have proposed audio paper as an alternative to the research article, suggesting that it affords a more situated and affective understanding of language and resonance of sound [Nancy 2007]. Collectively, propositions such as the above argue that drawing on methods and publication formats that are foreign to traditional academic reporting allow for an engagement with topics that are themselves foreign or difficult to articulate (as argued in [Amsertdam and Eck 2019]). Based on our collaboration with Katrine Meldgaard Kjær, I extend this argument of methodological disruption to development of a new critical language and vocabulary for analyzing code and programming.

Indeed, the mobilization of creative forms is not foreign to software studies and critical code studies scholarship. Relevantly, prior work has examined the relationship between code and poetry [Berry 2011];[Cox and McLean 2012] [Cox and McLean 2012] (see also [Heisler 2020];[Marino 2013]). We found that reading code aloud involves a physical, performative, embodied act of reciting code as a form of poetry or spoken word practice; following conventions of the *tidyverse* suite for the R programming language, individual lines perform a small action, and the lines are joined by the token `%>%` from the *magrittr* package. While reading the segment in Example 1 aloud, these symbols were vocalized as a repeated utterings "percent greater-than percent", giving the reading a meter. Its pulsing tempo lead my speaking body to move slightly back and forth, animating also the torso, arms and hands in an embarrassing sequence of groping movement hopelessly unrecognizable as a programming activity, but with a very salient experience in the lived body interacting with a text.

Reading code aloud thus moves analyzing code and programming as a practice into an arena that is more akin to storytelling. It explicitly encourages researchers to reflect on how they ascribe meaning to and react to hearing and speaking code in a new format. Reflexive and autoethnographic element of the approach connects code as text to lived, embodied experience, as well as to each participants' sensemaking processes around code. In our conversations and reading of code, this approach enables us to analyze how we each interpreted the meaning of the specific pieces of code that had been used in the project, but also on how hearing it being read aloud as such affected us. This became clear, for example, when I shared how hearing code being read aloud in adult work context prompted a memory from my youth:

"I have also like personal memories of reading code listings aloud from old magazines to my brother, who would type them on the computer we only had one computer and there was no internet or anything like that to download programs so they were printed in magazines and then one person would read them and then the other person would type, so this be like a personal, a too personal story, maybe you don't want to hear this but yeah, is this what I have to say? Yes, maybe."

Hearing code being read aloud and allowing for a space of reflection and resonance about those readings made us think back to our early memories of coding as a collaborative practice, but also allowed us to reflect on the different traditions that we work with code from now. With this, the questions of how personal positions and biographies were also embedded in larger structures became a central theme in our conversations. Here, the method of reading code aloud also borrows from feminist traditions for exploring and appreciating how personal experiences are a part of larger structural issues – the well-known motto "the personal is political" (see also [Rogan and Budgeon 2018]). Reading and listening affords critical intervention into code which is different from currently trending critiques of algorithmic power and "explanations" of their constitutive source code which seek to identify singular meanings of code. Indeed, it re-articulates explanation in relation to code to be a reflexive practice. With this, it is a method for a contextual and culturally situated reflection of the ways in which code is subject to cultural and social meaning-making processes.

De/re-familiarizing: reading aloud as a development of common language

about code

“Some of them are known words, words I feel like I’ve never not known, words like collect, select, pivot, tally.”

The transmedial gesture of abandoning the screen and the visual experience for reading code aloud in an alternative resonant sensory register enables a powerful de-familiarization from the usual ways we encounter source code. This de-familiarization opens the opportunity to grasp code as an object from a distance, which in turn can enable developing a new, common analytical language to unpacking code as a cultural object. In particular, we found this de-familiarization is powerful for developing a new, shared language for code across disciplinary divides, as its focus on turning away from explanation of code did work towards undoing the epistemic authority derived from the act of programming itself. While not always possible or even preferable, we argue that methods which allow the authors of code to participate in the interpretive work of code are vital for richer, shared understanding of software, and can even help repair alienating and historically false dichotomies of “them” as the othered programmers and “us” as the users [Mahoney 2005].

15

[Bronstein 2003] argues that success of interdisciplinary collaboration as such fundamentally depends on collective ownership of common research goals, as well as having to ability to ongoingly reflect on the process of collaboration. Likewise, Petrie [Petrie 1976] stresses the importance of the developing a common language in order to be able to articulate these common research goals. The adoption of a new, common language, however, does not replace the “old” scholarly language or disciplinarity, but rather adds to it and extends it. Together, the old and new create what may be understood as a kind of scholarly multimodality, where a more open-ended interrogatory approach necessarily replaces a declaratory approach [Tatum and Schumacher 2001]. In later research, the development of a common language has been articulated as central to establishing and sustaining so-called interdisciplinary “trading zones” [Collins et al. 2007]; [Fickers and van der Heijden 2020]. The development of these shared languages are always unique, and articulation processes local and situated. The typical division between “tool provider” and “issue expert” in Digital Humanities and social sciences collaborations, however, makes it difficult to develop these common languages, so much so that that these collaborations tend to be marked by “mutual incomprehension” [Bartscherer and Coover 2011].

16

Take the line `2 select(crawl_year, domain) %>%` from Example 1: a decision is to be made whether the “,” is vocalized as a comma, a pause, or something else. I stumbled on the line `12 mutate(rank = rank(desc(n), ties.method = 'first')) %>%` hesitating whether to read it left to right, or inside out following the execution order in R, all the while coming to doubt whether this line ought to be refactored altogether. Similarly, while the literal reading of the conjunction `%>%` as “percent greater-than percent” felt awkward at start, it gradually over a number of repetitions began to give an emphasized cadence and meter to the act of reading. Language construction decisions such as these have an immediate effect on the experience of reading and hearing the code, attuning both the reader and hearer to the intersubjective situation in which the reading takes place, unfamiliar to both. A central point here at that there is no authoritative or right way to read or hear code, and instead a local language comes to be improvised together. In this way, reading code aloud creates an engagement that has the potential to subvert or at least challenge not only disciplinary, but also embodied and gendered authority over programming.

17

Undoubtedly some code yields itself better to reading aloud than other code. For example, Fortran’s similarity to mathematical notation, heavy use of punctuation in the C family of languages, and Scratch’s node-based visuals them unwieldy to speak or listen. As an illustrative case, the esoteric programming language Orca by Rekka Bellum and Devine Lu Linvega might be near impossible to vocalize and hear due to its spatial layout in two dimensions – an irony since Orca is actively used to produce sound with electronic music instruments such as synthesizers Orca.

18

In our process of reading code aloud and responding aloud by recording and exchanging voice messages, Katrine found that hearing code aloud and away from the screen caused some words to be newly recognizable, while others remained a mystery. The process of turning away from the screen and towards talking about the custom code we used in the project as an object in and of itself allowed for more space to articulate our individual understandings of this code, as well as where these differ. In this, there was more space to reflect on their significance in the project and her sense-

19

making process around them. I myself, on the other hand, was confronted with how difficult it was to remove myself from an impulse to “explain” code to the discussion partner from the role of “tool provider”; a role that supported the division of labor that we were actively trying to undo. As I reflected in the beginning of our recording exchange:

"I feel like my early instinct is to like to explain this code, like start from the beginning and walk through, like, what its function is and what it does and what it does to what and what the outcome is. And what is the let's say, the connection between the inputs and outputs in a way, but it's not really like this kind of explaining, of course, of course, that has received a lot of attention recently. But I think there's more more to it."

While the role division of “issue expert” and “tool provider” was so deeply ingrown in us that it did not vanish when moving to audio, the de-familiarization that reading code aloud did make it salient, and allowed identifying and articulating these roles, as well as our impulses to step into them as they interpellate us [Althusser 1971]. This reflection was enabled by our explicit design for the method not to be about the instrumental use of code in the research, or attempt to familiarize everyone involved to evaluate, modify or maintain the code (see [Ojala 2021] and [Ojala and Cohn 2023] on maintenance as practice). Rather, what we were interested in was exploring code’s meaning to us and the connotations, associations and reflections of reading code and hearing code being read aloud provoked in us, without too much focus on the exact process the computer was instructed to do with any individual segment of the code. De-familiarization of code by reading it aloud, in other words, allowed for explicit reflection on the ways in which we were each situated in relation to our understanding of code, and our roles in relation to this particular custom code, and, importantly, what this meant for how code can be understood and conceptualized together. The distinct positions of issue expert and tool provider were backgrounded. What we had in common rose to the foreground, namely the research project as a shared task, and the powerful capacities to speak and listen, to associate, and to relate. One finding that came forth was our collective confrontation with the dominance of verbs in the pieces of code that were at the center of our research project. This was something we had not realized earlier in our project:

"(...) these words seem to be mostly names of actions. But here in this case, or in this research, these actions are also familiar in other kind of research as you definitely point out, but now hearing them say aloud, again, of course, I said them aloud earlier, but hearing them aloud, again, makes very salient how very much they are verbs."

As one of us notes in the above transcript excerpt, hearing the pieces of code being read aloud drew our attention to the linguistic composition of the pieces of code, and how the verbs in the code made them very focused on actions. Did thinking in motion, using handsfree headset away from the computer screen while walking, breathing, attending to the ebbs and flows of surrounding bicycle traffic contribute to observation of their activeness? Did the words of crawling (Example 1, lines 2, 3, 6, 8, 9 and 10) and mutating (line 7) acquire a more vivid meaning in the park rather than at the ergonomically standardized electronic standing desk at the university campus?

In her reflections, cited at the beginning of this section, Katrine rightly noted the prominence of “generic” verbs such as select (Example 1, lines 2 and 9) and tally (line 4) and collect (line 5) after initially having heard the code being read aloud. In hearing these words aloud, they were for her re-connected with the worldly life from which they were borrowed in the first place, to symbolized actions on virtual servers, CSV files and VPN connections (see Example 1 and excerpt at the beginning of this section). Hearing the piece of code being read aloud removed from other context also caused her to reflect on the notable absence of the word “medicinal cannabis” in a project on medicinal cannabis:

"But I noticed how, as you also remark, the thing that, the theme of the thing that we were studying – medicinal cannabis – is only, it's in there in the introduction (...) it becomes absent for the rest of the piece of code. So this makes me think about how things are sorted with these same mechanisms, like could this just been any project?"

This observation lead us to articulate how our code named abstract, generic actions (e.g. “select”, “tally”, “collect”) rather than picked up the immanence of the specific research topic and the computation infrastructure at hand. Computation’s tendency toward the generalization and abstraction is typical, as it distances from application domains,

not unlike how mathematics and symbolic logic formalize from domains such natural Sciences, geometry, music, dialectics and rhetorics [Oregon 2017];[Ribes 2019]. Together, each taking turns to vocalize our own understandings of these words, we could examine the code's loosening relationship with the content of the research project, namely the discursive constructions of medicinal cannabis and debates around it. This exchange highlighted that exactly same segment of custom code would have made sense in any other project: the mechanics of the analysis were actually independent from the topic of analysis itself; we had unintendedly crafted a commodity. Moving from screen to audio involved a kind of de-familiarization of the pieces of code that we had been programming and working with for a longer period of time, and provoked new conversations both about code as analytical tool and about the text of code itself.

De/re-contextualizing: reading aloud for situated, affective analysis of code

Reading code aloud enables a de-contextualization from the screen and roles of “analyst” and “programmer”, which, in turn, opens new space for a re-contextualizing of code and its interpretations as coming from those who work with and on it. Drawing on research from feminist technoscience, we propose that reading code aloud allows for a more clearly situated approach to studying code and its significance. Donna Haraway [Haraway 1991](1991) argues that all knowledge is situated, and that the generation of knowledge is therefore not a product of a neutral “God’s eye” perspective on a given phenomenon, but rather comes from somewhere, and some-body. In short, this entails that what we as researchers can see and know is shaped by both the tools that we entangle with to study a given phenomenon, as well as our own lived experiences, which informs what we (can) see; from this follows that knowledge is also contextual and embodied. The practices involved in big data analysis, however, have been associated with the promise of neutral, “bias-free”, disembodied knowledge generation because they involve computational as opposed to qualitative modes of analysis [Baker 2012]. This, in turn, has been critiqued by scholarship from critical data studies, who have argued that big data practices are indeed situated, and, by extension, political [Iliadis and Russo 2016]. Indeed, there has been research on the situatedness of Digital Humanities work in terms of location and space ([Oiva and Pawlicka-Deger 2020]) and discipline-specific knowledge ([Van Gorp, Gorp, and Bron 2019]), and feminist and intersectional research has explored the relationship between embodied positions, discriminatory practices and computing (see e.g. [Benjamin 2019];[Hicks 2017]. However, there is still much to explore in relation to how to unpack the situatedness and embodiment of the researcher themselves in the practices that go into big data work, including programming Developing methods that can engage with the situatedness and embodied character of coding practices, then, is crucial to develop understandings of the implications of such work. This, we suggest, can be a key contribution from critical code studies.

One way in which reading code aloud made clear the embodied and situated character of code was that it allowed for a space to engage with the affective or emotional character of code and programming. Evalyn et al. have written about the affective community that coded signs such as emojis can create in the digital humanities ([Evalyn et al. 2020]), and when reading code aloud we found that this affective potential also applied to the code of our project. At first sight, the situatedness of the code presented in this article obviously related to the two people conducting the research. This context became differently tangible, however, as we turned away from the screen and the instructional explanations of the code we had programmed, and instead shared recordings of our voices associating with the code more subjectively; such as when we shared words which sounded known or unknown to us, or memories of earlier programming situations. The intersubjective nature of the affective or emotional aspects of discussing code – as element that had previously been less pronounced in our discussions of code – also became salient as we imagined each other’s presence while recording and listening, and while anticipating one another’s response. For instance, we quickly came to establish a mode of communication where we started by thanking or complimenting the other for their previous reflections, and expressing joy at the process we were in the middle of. Recording reflections and sending them to each other, then, opened a space for a different kind of relationship between us and code, one that had a more personal, even intimate character. An aspect of the situatedness and embodied character of coding work that became apparent through our reflections in this space was the emotional character of programming itself. Rather than being a neutrally instrumental or disaffected practice, code and coding was associated in our reflections with emotions, such as frustration, vertigo, care, surprise, and even disappointment. When recounting the work of writing the attached code (see Example 1) for the medicinal cannabis project, I spontaneously reflected on how disappointment was a part of his process:

24

25

"But yeah, one word I remember. But this is experienced based, but I remember one word that was very hard to do was this word of collect. And I think I spent an entire day on it. (...) I remember, it was very disappointing to include this word collect, it was like a sort of a recognition of the materiality of this setup, and the infrastructure that we were working with, and that we were bound with, like a disappointing and like unnecessary word or name of an action."

The object of this frustration was related to the cloud-based computational environment provided to us by Danish Royal Library, which necessitated running the Spark computational engine on one server and the R session on another for the code to successfully run. To solve this, we used the *collect* verb of the *dplyr* grammar, which is part of the influential *tidyverse* suite of data science tools for the R language ([Wickham et al. 2019]), on line 5 of the provided excerpt (see Example 1). According to reference documentation, *collect* "forces computation of a database query" [Wickham et al. 2020]. Inclusion of this command, and the emotional reaction to it, was thus not justified by the data analytical task, but rather by the infrastructure we were using.

26

The tactical moves related to "collect", which frustratingly distracted from the data analytical task, are thus also important reminders of another aspect of the situated nature of code in the project, namely the material infrastructure in which its programming took place. The history of programming languages is characterized by an attempt to separate hardware from software and abstract away the machine [Chun 2011];[Mahoney 2005];[Marino 2020]. However, this move to de-situate has also been critiqued for not being possible as such; Wendy Chun has labeled the idea that source code is separable from the machines and from its execution as "sourcery" ([Chun 2011]), and as Mackenzie has shown for Java, such virtuality is a practical but also fragile achievement [Mackenzie 2006]. The method of reading code aloud shined a light on the fictional nature of this separation on multiple occasions—for example as confusion and disappointment about the inevitable use of the "collect" verb, although the code ran and produced expected results; as this example demonstrates, the source code is not separated from its context of execution by "hardware" or "wetware" ([Rucker 1988]) nor is it the singular source of final meaning. By opening a space for explicit reflection on our experiences of code, we were thus able to build understanding how code and infrastructure co-create the experience of programming and research in not just functional, but also lively and affective assemblages.

27

Reading code aloud emphasizes a reflexive and contextual approach to code and programming practices. The embodied and situated nature of interacting with code is often invisible in analyses of code. However, engaging with a method for an analysis of code that directs attention away from the screen and toward the lived and resonant experience of working with code offers the opportunity to shine light on the situated nature of the production and interpretation of code. Following scholars such as Haraway, all knowledge is situated and comes from somewhere, and accounting for this is a key aspect of understanding the power dynamics of knowledge production as such. With this, developing methods for engaging with the relationship between embodiment, code and programming is central for an understanding of the social and cultural construction, reception and understandings of code.

28

Outro: re-engaging code

In this text I have presented and reflected on my and my colleague's Katrine Meldgaard Kjær's use of the method of reading code aloud using recorded voice messages over a few weeks time. Taking a point of departure in experiments with reading a relatively small piece of software, in this case 11 lines of code of our own creation (see Example 1), I have argued that moving from screen to audio can be generative for tracing out connotations, interpretations and new questions about code. Our proposed method, developed with Katrine, hinges on three central moves. First, it was something we did together, but also separately: the recoding [*sic.*] ^[1] of each of our reflections prompted different listening and speaking techniques and capabilities than in more conventional meetings. Second, we used the method to work against the division of "programmer" and "researcher", actively moving out of the space of the screen which normally sustains these roles. Third, we used the method to facilitate a disengagement with instructional, functional explanations of code, favoring instead a dialectical and open-ended interpretative mode; in this way, we moved from understanding "programmer" and "code" as nouns to approaching them as verbs (see [Chun 2011]).

29

We found that moving into audio as the realm for analysis of code activated a number of de- and re-familiarizing

30

movements. Firstly it de-explained the code as a useful, expertly knowable function from inputs to outputs and re-explained it as a living and interpretable text. Second, it de-familiarized code from being a language on screen which divided “programmer” and “analyst” into two separate roles, and re-familiarized code as something which we shared a common experience of and could create a common language about. Finally, it de-contextualized code from the idea of being disembodied, neutral and unsituated, and allowed re-contextualizing code as an embodied and affective practice, as well as created a space to appreciate how this practice involves and is shaped by infrastructure and relationships between hardware and software, as well as our relationships to those. Collectively such de- and re-familiarizing movements can allow a mode of critical engagement with and about code that takes its point of departure not just from a functional, but also a cultural, social and personal understanding and analysis of self-programmed as well as received code. With this, I argue that methodologies which work intentionally with processes of de- and re-familiarization provide a fruitful point of intervention for critical code studies work in the future. In proposing reading code aloud as one such method, I hope to hear from future scholarship that enables contextual, situated, reflexive and critical analysis of code and its text and context.

Acknowledgements

I express gratitude and respect for my peer Katrine Meldgaard Kjær for her wise and powerful companionship. While this article is officially single authored, none of this would exist without this collaboration.

31

Notes

[1] Originally intended to be “recorded” rather than “recoded” in our draft. As the typo was appropriate, we decided to keep it.

Works Cited

Abbate 2012 Abbate, J. (2012) *Recoding Gender*. MIT Press.

Abbate 2021 Abbate, J. (2021) “Coding is not empowerment”, in: Mullaney T, Peters B, Hicks M, et al. (eds) *Your computer is on fire*. MIT Press. Available at: <https://doi.org/10.7551/mitpress/10993.003.0018>.

Althusser 1971 Althusser, L. (1971) Ideology and Ideological State Apparatuses (Notes towards an Investigation). In: *Lenin and philosophy and other essays*, pp. 121–176. Available at: <http://www.marxists.org/reference/archive/althusser/1970/ideology.htm>.

Amsertdam and Eck 2019 Amsterdam, N. van and Eck, D. van (2019) “In the flesh: a poetic inquiry into how fat female employees manage weight-related stigma”, *Culture and Organization* 25(4): 300–316. DOI: 10.1080/14759551.2019.1598999.

Baker 2012 Baker, P. (2012) “Acceptable bias? Using corpus linguistics methods with critical discourse analysis”, *Critical Discourse Studies* 9(3): 247–256. DOI: 10.1080/17405904.2012.688297.

Baker et al. 2008 Baker, P., Gabrielatos, C., KhosraviNik, M. et al. (2008) “A useful methodological synergy? Combining critical discourse analysis and corpus linguistics to examine discourses of refugees and asylum seekers in the UK press”, *Discourse & Society* 19(3): 273–306. DOI: 10.1177/0957926508088962.

Barad 2007 Barad, K. (2007) *Meeting the universe halfway: Quantum physics and the entanglement of matter and meaning*. Duke University Press.

Bartscherer and Coover 2011 Bartscherer, T. and Coover, R. (2011) *Switching Codes: Thinking Through Digital Technology in the Humanities and the Arts*. Chicago: University of Chicago Press. Available at: <https://press.uchicago.edu/ucp/books/book/chicago/S/bo6027946.html> (accessed 17 March 2021).

Benjamin 2019 Benjamin, R. (2019) *Race after technology: Abolitionist tools for the new Jim code*. Wiley.

Berry 2011 Berry, D. (2011) *The philosophy of software. Code and mediation in the digital age*. Palgrave Macmillan.

Bornakke and Due 2018 Bornakke, T. and Due, B.L. (2018) “Big–thick blending: A method for mixing analytical insights from big and thick data sources”, *Big Data & Society* 5(1). DOI: 10.1177/2053951718765026.

Bradley et al. 2018 Bradley, A.J., El-Assady, M., Coles, K., et al. (2018) “Visualization and the digital humanities”, *IEEE Computer Graphics and Applications* 38(6): 26–38. DOI: 10.1109/MCG.2018.2878900.

- Bronstein 2003** Bronstein, L. (2003) "A model for interdisciplinary collaboration", *Social work* 48(3): 297–306.
- Buxton 2007** Buxton, B. (2007) *Sketching user experiences: Getting the design right and the right design*. 1st edition. Amsterdam: Morgan Kaufmann.
- Ceglowski 1997** Ceglowski, D. (1997) "That's a Good Story, But Is It Really Research?", *Qualitative Inquiry* 3(2): 188–201. DOI: 10.1177/107780049700300203.
- Chun 2011** Chun, W.H.K. (2011) *Programmed visions. Software and memory*. MIT Press. Available at: <https://mitpress.mit.edu/books/programmed-visions> (accessed 10 March 2021).
- Collins et al. 2007** Collins, H., Evans, R., and Gorman, M. (2007) "Trading zones and interactional expertise", *Studies in History and Philosophy of Science Part A* 38(4). Case studies of expertise and experience: 657–666. DOI: 10.1016/j.shpsa.2007.09.003.
- Cox and McLean 2012** Cox, G. and McLean, A. (2012) *Speaking code. Coding as aesthetic and political expression*. MIT Press.
- Diakopoulos 2015** Diakopoulos, N. (2015) "Algorithmic accountability. Journalistic investigation of computational power structures", *Digital Journalism* 3(3). Routledge: 398–415. DOI: 10.1080/21670811.2014.976411.
- Elbow 1997** Elbow, P. (1997) "Collage: Your cheatin' art", *Writing on the Edge* 9(1): 26–40.
- Ellis 2004** Ellis, C. (2004) *The autoethnographic I*. Walnut Creek, California: AltaMira.
- Ensmenger 2010** Ensmenger, N.L. (2010) *The computer boys take over: Computers, programmers, and the politics of technical expertise*. History of Computing. Cambridge, MA, USA: MIT Press.
- Evalyn et al. 2020** Evalyn, L., Henderson, C.E.M., King, J. et al. (2020) "One loveheart at a time: The language of emoji and the building of affective community in the digital medieval studies environment", *Digital Humanities Quarterly* 14(3).
- Fickers and van der Heijden 2020** Fickers, A. and Heijden, T. van der (2020) "Inside the trading zone: Thinkering in a digital history lab", *Digital Humanities Quarterly* 014(3).
- Fürsich 2009** Fürsich, E. (2009) "In defense of textual analysis", *Journalism Studies* 10(2): 238–252. DOI: 10.1080/14616700802374050.
- Graban et al. 2019** Graban, T.S., Marty, P., Romano, A. et al. (2019) "Introduction: Questioning collaboration, labor, and visibility in digital humanities research", *Digital Humanities Quarterly* 013(2).
- Haraway 1991** Haraway, D.J. (1991) A Cyborg Manifesto: Science, Technology, and Socialist-Feminism in the Late Twentieth Century. In: *Simians, Cyborgs, and Women: The Reinvention of Nature*. Routledge, pp. 149–181.
- Heisler 2020** Heisler, E. (2020) "Winnie soon, time, code, and poetry", *Asymptote*. Available at: <https://www.asymptotejournal.com/visual/winnie-soon-time-code-and-poetry/> (accessed 5 February 2021).
- Hicks 2013** Hicks, M. (2013) "De-programming the history of computing [Think Piece]", *IEEE Annals of the History of Computing* 35(1): 88–88. DOI: 10.1109/MAHC.2013.3.
- Hicks 2017** Hicks, M. (2017) *Programmed inequality: How Britain discarded women technologists and lost its edge in computing*, History of Computing. Cambridge, MA, USA: MIT Press.
- Hunt and Thomas 1999** Hunt A and Thomas D (1999) *The Pragmatic Programmer: From Journeyman to Master*. 1st edition. Reading, Mass: Addison-Wesley Professional.
- Iliadis and Russo 2016** Iliadis, A. and Russo, F. (2016) "Critical data studies: An introduction", *Big Data & Society* 3(2): 2053951716674238. DOI: 10.1177/2053951716674238.
- Kearns and Roth 2019** Kearns, M. and Roth, A. (2019) *The ethical algorithm: The science of socially aware algorithm Design*. New York: Oxford University Press.
- Kirk 2019** Kirk, A. (2019) *Data visualisation: A handbook for data driven design*. 2nd edition. S.I.: SAGE Publications Ltd.
- Knuth 1984** Knuth, D.E. (1984) "Literate programming", *The Computer Journal* 27(2): 97–111. DOI: 10.1093/comjnl/27.2.97.
- Krogh Groth and Samson 2016** Krogh Groth, S. and Samson, K. (2016) "Audio papers – a manifesto", *Seismograf* 16(01). DOI: 10.48233/seismograf1601.

- Mackenzie 2006** Mackenzie, A. (2006) *Cutting code. Software and sociality*. Digital Formations. Peter Lang. Available at: <https://www.peterlang.com/view/title/58468> (accessed 6 July 2020).
- Mahoney 2005** Mahoney, M.S. (2005) "The histories of computing(s)", *Interdisciplinary Science Reviews* 30(2). Taylor & Francis: 119–135. DOI: 10.1179/030801805X25927.
- Marino 2013** Marino, M. (2013) "Code as ritualized poetry: The tactics of the transborder immigrant tool", *Digital Humanities Quarterly* 007(1).
- Marino 2014** Marino, M. (2014) "Field report for critical code studies", 2014, *Computational Culture* (4). Available at: <http://computationalculture.net/field-report-for-critical-code-studies-2014/> (accessed 18 March 2021).
- Marino 2020** Marino, M. (2020) *Critical code studies*. MIT Press.
- Nancy 2007** Nancy, J.-L. (2007) *Listening* (tran. C Mandell). Fordham University Press.
- O'Neil 2016** O'Neil, C. (2016) *Weapons of math destruction: How big data increases inequality and threatens democracy*. 1st edition. New York: Crown.
- Oiva and Pawlicka-Deger 2020** Oiva, M. and Pawlicka-Deger, U. (2020) "Lab and slack. Situated research practices in digital humanities - Introduction to the DHQ special issue", *Digital Humanities Quarterly* 14(3).
- Ojala 2021** Ojala, M. (2021) Maintain-ability : A thesis on life alongside computer software. Available at: <https://urn.fi/URN:NBN:fi:tuni-202202031820>.
- Ojala and Cohn 2023** Ojala, M. and Cohn, M. (2023) "Software maintenance as materialization of common knowledge", *Engaging Science, Technology and Society*, 9(3). Available at: <https://doi.org/10.17351/ests2023.1325>.
- Oregan 2017** O'Regan, G. (2017) "A short history of logic", in: O'Regan G (ed.) *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications*. Undergraduate Topics in Computer Science. Cham: Springer International Publishing, pp. 93–108. DOI: 10.1007/978-3-319-64021-1_5.
- Petrie 1976** Petrie, H.G. (1976) "Do you see what I see? The epistemology of interdisciplinary inquiry", *Educational Researcher* 5(2). American Educational Research Association: 9–15. DOI: 10.3102/0013189X005002009.
- R Core Team 2020** R Core Team (2020) *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Available at: <https://www.R-project.org/>.
- RStudio Team 2020** RStudio Team (2020) *RStudio: Integrated development environment for R*. Boston, MA: RStudio, PBC. Available at: <http://www.rstudio.com/>.
- Ribes 2019** Ribes, D. (2019) "How I learned what a domain was", *Proceedings of the ACM on Human-Computer Interaction* 3(CSCW): 38:1-38:12. DOI: 10.1145/3359140.
- Ribes et al. 2019** Ribes, D., Hoffman, AS., Slota, S.C. et al. (2019) The logic of domains. *Social Studies of Science* 49(3). SAGE Publications Ltd: 281–309. DOI: 10.1177/0306312719849709.
- Rogan and Budgeon 2018** Rogan, F. and Budgeon, S. (2018) "The personal is political: Assessing feminist fundamentals in the digital age", *Social Sciences* 7(8). 8. Multidisciplinary Digital Publishing Institute: 132. DOI: 10.3390/socsci7080132.
- Rucker 1988** Rucker, R. (1988) *Wetware*. Avon Books."
- Schwandt 2007** Schwandt, T.A. (2007) Crisis of Representation -. *SAGE Research Methods*. 3rd edition. Available at: <https://methods.sagepub.com/reference/the-sage-dictionary-of-qualitative-inquiry/n55.xml> (accessed 29 January 2021).
- Swidan 2019** Swidan, A. and Hermans, F. (2019) "The effect of reading code aloud on comprehension: An empirical study with school students", in: *Proceedings of the ACM Conference on Global Computing Education*, New York, NY, USA, 9 May 2019, pp. 178–184. CompEd '19. Association for Computing Machinery. DOI: 10.1145/3300115.3309504.
- Tatum and Schumacher 2001** Tatum, J.S. and Schumacher, J.A. (2001) "Knowledge: Repositioning, redirection, and reform", *Organization* 8(2): 425–432. DOI: 10.1177/1350508401082027.
- Terras et al. 2013** Terras, M., Nyhan, J., and Vanhoutte, E. (2013) *Defining digital humanities: A reader*. Routledge. Available at: <https://www.routledge.com/Defining-Digital-Humanities-A-Reader/Terras-Nyhan-Vanhoutte/p/book/9781409469636> (accessed 20 June 2020).
- Van Gorp, Gorp, and Bron 2019** Van Gorp, J., Gorp, J.V., and Bron, M. (2019) "Building bridges: Collaboration between

computer sciences and media studies in a television archive project", *Digital Humanities Quarterly* 013(3).

Wang et al. 2017 Wang, Q., Coemans, S., Siegesmund, R. et al. (2017) "Arts-based methods in socially engaged research practice: A classification framework", *Art/Research International: A Transdisciplinary Journal* 2(2). 2: 5–39. DOI: 10.18432/R26G8P.

Wickham et al. 2019 Wickham, H., Averick, M., Bryan, J. et al. (2019) "Welcome to the tidyverse", *Journal of Open Source Software* 4(43): 1686. DOI: 10.21105/joss.01686.

Wickham et al. 2020 Wickham, H., François, R., Henry, L. et al. (2020) *Dplyr*. tidyverse, R.

Xie et al. 2018 Xie, Y., Allaire, J.J. and Golemund, G. (2018) *R markdown: The definitive guide*. Boca Raton, Florida: Chapman and Hall/CRC. Available at: <https://bookdown.org/yihui/rmarkdown>.

Zaharia et al. 2026 Zaharia, M., Xin, R.S., Wendell, P. et al. (2016) "Apache spark: a unified engine for big data processing", *Communications of the ACM* 59(11): 56–65. DOI: 10.1145/2934664.

Ziewitz 2016 Ziewitz, M. (2016) "Governing algorithms: Myth, mess, and methods", *Science, Technology, & Human Values* 41(1). SAGE Publications Inc: 3–16. DOI: 10.1177/0162243915608948.

Recommendations

DHQ is testing out three new article recommendation methods! Please explore the links below to find articles that are related in different ways to the one you just read. We are interested in how these methods work for readers—if you would like to share feedback with us, please complete our short evaluation survey. You can also visit our documentation for these recommendation methods to learn more.

SPECTER Recommendations

Below are article recommendations generated by the SPECTER model:

1. Introduction: Reading Code in the Age of AI, 2026, Jeremy Douglass, University of California, Santa Barbara; Mark C. Marino, University of Southern California
2. Defactoring Pace of Change, 2026, Matt Burton, University of Pittsburgh; Joris van Zundert, Huygens Institute of the Royal Netherlands Academy of Arts and Sciences
3. Tracing "Toxicity" Through Code: Towards a Method of Explainability and Interpretability in Software, 2023, David M. Berry, University of Sussex
4. How to Do Things with Deep Learning Code, 2023, Minh Hua, Johns Hopkins University; Rita Raley, University of California, Santa Barbara
5. Introduction: Situating Critical Code Studies in the Digital Humanities, 2023, Mark C. Marino, University of Southern California; Jeremy Douglass, University of California, Santa Barbara

DHQ Keyword Recommendations

Below are article recommendations generated by DHQ Keywords:

1. The Productive Unease of 21st-century Digital Scholarship, 2009, Julia Flanders, Brown University
2. Service-Oriented Software in the Humanities: A Software Engineering Perspective, 2010, Nicolas Gold, King's College London, Department of Computer Science
3. A review of Brett D. Hirsch (Ed.)'s *Digital Humanities: Pedagogy: Practices, Principles, and Politics*, 2014, Erik Shell, University of Maryland
4. Potentialités et difficultés d'un projet en humanités numériques (DH) : confrontation aux outils et réorientations de recherche, 2018, Christelle Cocco, Université de Lausanne; Grégory Dessart, Université de Lausanne; Olga Serbaeva, Universités de Lausanne et de Zürich; Pierre-Yves Brandt, Université de Lausanne; Dominique Vinck, Université de Lausanne; Frédéric Darbellay, Université de Genève
5. An Agent-based Model for the Humanities, 2013, Belinda Roman, St. Mary's University

TF-IDF Recommendations

Below are article recommendations generated by the TF-IDF Model:

1. Audiovisualities out of Annotation: Three Case Studies in Teaching Digital Annotation with Mediate, 2021, Joel Burges, University of Rochester; Solvegia Armoskaite, University of Rochester; Tiamat Fox, University of Rochester; Darren Mueller, University of Rochester; Joshua Romphf, University of Rochester; Emily Sherwood, University of Rochester; Madeline Ullrich, University of Rochester
2. Introduction: Reading Code in the Age of AI, 2026, Jeremy Douglass, University of California, Santa Barbara;

Mark C. Marino, University of Southern California

3. Code as Ritualized Poetry: The Tactics of the Transborder Immigrant Tool, 2013, Mark C. Marino, University of Southern California
4. Stitching the Fragmented: Feminist Maker Pedagogy and Immersive Technologies for Cultural Learning, 2022, Mélanie Péron, University of Pennsylvania; Meaghan Moody, University of Rochester; Vickie Karasic, Bryn Mawr College
5. Criminal Code: Procedural Logic and Rhetorical Excess in Videogames, 2013, Mark L. Sample, George Mason University



This work is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License.