Summary - I

02.01.2020

This document covers the research work that I've done during AY 2019/2020. The work I've done in this first semester has been focused on two main parts. The first part has consisted in reading the current state of academic literature which deals closest to my topic (source code aesthetic/creativity), in order to identifying existing trends and contributions, as well as inform perspectives for my own research. The second part has resulted in the collection of primary sources, and in organizing these in preliminary categories. These two parts, along with further readings in literary theory in general has allowed me to highlight a more precise angle for my proposed research, along the lines of *the role of programming languages as material in source code poetics*.

Recent literature about the field of source code aesthetics, broadly understood as any inquiries centering around the writing, the reading and the distributing of source code, covers fields as diverse as literature (Hayles, Black, Sondheim), science and technology studies (MacKenzie), humanities (Paloque-Berges), software studies (Montfort et. al., Cox and McLean) and philosophy (Cramer). In particular, each of these monographs (and in some cases, catalog articles or book chapters) tend to engage with code at a level beyond that of the broad, diffuse concept. Source code excerpts, programming environments and languages form a part of each of these works as primary sources and are considered as *texts* to be read and examined closely, and in that sense all offer the same foray into code-as-text as my work intends to.

I will first provide an overview of each contribution organized by fields, and then elaborate on how these contributions informed the further definition of my research approach on programming languages. This is followed by a brief overview of the corpus of primary sources which I have selected so far, and concludes with some notes and questions about next steps.

Literature review

While most sources, because of their dealing with source code as text, incorporate some aspect of literary theory and criticism, the works of N. Katherine Hayles, Maurice J. Black and Alan Sondheim rely on it as their principal perspective. Black, in his PhD dissertation *The Art of Code* (University of Pennsylvania, 2002) initiates the idea of a cross between programming and literature, and hypothesizes that writing source code is an act that is closer to modernism than postmodernism, since it relies on ideas of authorship, formal linguistic systems, and somewhat self-reference. The intent here is to show how code functions with its own aesthetic, one which is distinct and yet closely related to a literary aesthetic. After highlighting how the socio-political structures of computing since the 1950s (e.g. free software vs. closed-source, academic development) have affected the constitution of a the idea of a code aesthetic both in professional and amateur programmers, Black moves towards the examination of code practices as aesthetic practices. Here, Black limits himself to the presentation of coding practices insofar as they are identified and referred to as aesthetic practices, but exclusively through a social definition of aesthetic -how groups and individuals refer to their own practice- without going into further depth into the analytical questions of *what is it* that makes source codes and literary texts so similar, and yet so different to human languages. He mentions the practice of source code poetry, as perl poetry, and at the same time notes that there hasn't be a rigorous literary criticism of source code poetry, an endeavor he briefly touches upon. Finally, the last section of the dissertation engages in an analysis of *Finnegan's Wake* through the lens of Object-Oriented Programming (OOP) paradigm, hypothesizing that coding practices and models can provide useful heuristics for the analysis of classical literature.

At this point, it seems that Black operates this study in only one of two directions: by equating programming and literature, he assumes that it is possible to write about literature through the lens of source code. However, the object of his study has only been defined socially as a literary object, rather than formally, by looking at the language structures that form the basis of programming. As such, the actual analysis of source code with the help of formal literary theories is almost entirely side-stepped. In summary, Black provides a first study in code as a textual *object* and a textual *practice*, but falls short of establishing foundations of a study of code a formally aesthetic object, in particular working on the limits of what makes Perl poetry (or any other programming-language-based poetry, for that matter) different in its writing, reading and meaning-making than, say, natural-language poetry.

N. Katherine Hayles, in her book *My Mother Was A Computer: Digital Subjects and Literary Texts* (University of Chicago, 2005) temporarily removes code from its immediate social and historical situations, links it as a *worldview*, and attempts to qualify this worldview both in opposition to Saussure's *parole* and Derrida's *trace*, following cybernetics and media studies thinkers such as Friedrich Kittler and Mark B. Hansen. Specifically, she introduces her idea of the *Regime of Computation*, which relies on the conceptual specificities of code-based expression (among which: depth, dynamism, fragmentation, etc.). While the second part of the work, similarly to Black, focuses on the application of heuristics from the Regime of Computation to provide new readings of either classical as well as electronic literature. Source-code specific contributions are found in the first chapter, highlighting the way code recombines some traditional dialectics of literary theory, namely paradigmatic/syntagmatic, discreete/continuous, compilation/interpretation, and flat/stacked languages. The closest she comes to a close-reading is when she mentions that programming paradigms, such as OOP as opposed to procedural languages, or that programming languages, such as C++, would affect the structure and the meaning of programming texts, describing how the *syntax* of object-oriented programming reflects the syntax of human languages and therefore unlock source code as a literary text.

While Hayles provides the basis for a much deeper analysis of source code's formal literary properties, she also maintains that source code studies should keep in mind the ever-underlying materiality that this very source code relies on. She then locates this materiality in the embodiment of users and readers, along with Mark Hansen and Bruno Latour, considering the *wetware* along with the *hardware* or *software*. Beyond the brief acknowledgment that she has of the political and economical conditions of software development and their impact on electronic texts, she also stops short of considering specifically programming languages, and the material apparatuses which support them (documentation, architectures, compilers, tutorials, conferences and communities), as this material does exhibit some *family resemblances*. Such an approach of programming languages as material seems like a possible avenue for looking into the formal specificities of programming languages both between them and human languages, but also between one another (e.g. COBOL vs. C++ vs. JavaScript).

Alan Sondheim's essay *Codework* (American Book Review, 2001), as the introduction of the American Book Review issue dedicated to this specific form, provides another aspect of poetry that which integrates source code as a creole language emerging from the interplay of natural and machine languages. However, this specific aspect of literary work integrates the *surface* of code rather than its structure and therefore provides more insight in the anthropology of how humans represent code through speech, rather than representing speech through code (i.e. focusing first and foremost on the structural uniqueness of programming languages). This presents a somewhat postmodern view of programming languages, forcing them upon a relational, mutable conception of language as as series speech-acts, and leaving aside their structural and post-structural characteristics. Codework is essentially defined by its content and *milieu*, the networked environment in which emails, instant messages and delivery reports constitute the raw material. This material is then fragmented and re-inserted as a contaminating agent of human exchanges. This aspect of the interaction between code and speech only provides limited contribution in terms of my research since it focuses more on the *parole* than on the *langue*, and is limited to a somewhat narrow body of identified authors, mailing lists and works, *a priori* casting it as an established practice within the net.art movement, rather than as a potentiality for literature in general.

Also drawing on the relationship between speech and code, Geoff Cox and Alex Mclean develop in *Speaking Code: Coding as Aesthetic and Political Expression* (MIT Press, 2012) an interpretation of reading, writing and executing source code as a speech-act in the broader political sense. To this end, they start Arendt's approach of human activities in *The Human Condition*, and identify labor as a speech-act, from which it follows that coding is the practice of producing laboring speech-acts.

Source code is considered as a located, instantiated presence, a political position from which it can be understood as a semantic object affecting the multiple economic, social and discursive environments in which it lives. Focusing on speech particularly, this study doesn't quite address the syntactic specificities of codes, for example by looking at the use of loops, arrays, or other syntactical structures which could have very well fit within their overall argument. However, on a more formal level, the authors often illustrate the points they are working through, or begin developing those points, with snippets of code written by either McLean or established software artists. This shows the intention of engaging with details of source code by highlighting it through the book to support their argument, and taking a step away from the dangers of fetishizing code, or *sourcery*, to put it in Chun's terms. Of all the studies reviewed at this stage of my research, this is one of the most direct engagements with source code I have encountered, since it includes both deductive code (commenting existing source code) or inductive code (code written to act as an example to a point developed by the authors). In the end, I've found the limitations of that work to rely overly on the particular position of speech which, while illuminating, still side-steps the particular grammatical features of that speech, features such as Object design and verbosity which authors such as Tirell and Galloway explicitly shown as enabling acts of political expression and production.

Away from the cultural relevance of code as developed by Cox and McLean, Cramer focuses on the cultural history of computation, tying our contemporary fascination with source code into an older web of historical attempts at integrating combinatorial practices from Hebraic texts to Leibniz's universal languages. It is in this space between magic and logic that Cramer locates today's experiments in source code (i.e. source code poetry, esoteric languages and codeworks). By relocating it there, code is no longer just arbitrary symbols, or machine instructions but also ideal execution. Once formal execution is considered a cosmogonical force, it becomes synonymous with performative execution, through which it ties back to cultural practices throughout the ages. In this exploration, he extracts 5 axes along which to apprehend code-based works: totality/fragmentation, rationalization/occultation, hardware/software, syntax/semantics, artificial/natural language.

While all these axes overlap each other, it is the *syntax/semantics* axis which seems the most aligned with the current angle of my research. Indeed, I hypothesize that it is possible to touch upon these other thematical axes through the lens of syntax and semantics. Towards the end of the book, his development of the concept of speculative programming is also particularly fruitful as the attempt to become a figure of thought and reflection in theory and artistic practice. Speculative programming could be an interesting concept when articulated with some of the primary sources I've gathered (particularly regional programming languages, see below). Finally, Cramer states that *formalisms [...] have a cultural semantics of their own, even on the most primitive and basic level. With a cultural semantics, there inevitably is an aesthetics, subjectivity and politics in computing.* While these cultural semantics have been highlighted in multiple ways so far, how the formalisms themselves implement this culture and those semantics remain to be developed further.

Following literature and cultural studies, the last group of work is organized around sociology, science and technology studies (STS) and software studies.

Cutting Code: Software and Sociality (Peter Lang, 2006) by Adrian MacKenzie approaches software first through a problem of definition. The author establishes a relational ontology of software: it is defined in how it acts upon, and is being acted upon by existing external structures, from intellectual property frameworks to design philosophies in software architectures. Most of the book is devoted to finding software's place inside the broader world, and therefore only defines it by what it *does*, for a lack of defining it for what is *is*. His analysis of source code poetry focuses on famous Perl poems, Jodi's artworks and Alex McLean's *forkbomb.pl* and highlights the executability of code as its dominant feature, dismissing Perl poetry as "*a relatively innocuous and inconsequential activity*" which should really be considered as a gateway towards understanding the executability of code through its textuality, rather than stopping temporarily to examine textuality for itself, which he only addresses in passing. While software could indeed be a *patterning of social relations*, it would also be possible to explore these social relations as happening through linguistic means. In the end, MacKenzie's work helps putting into perspective the very issues of being able to even define software itself, and the definition he provides is indeed useful for further studies.

Camille Paloque-Berges published, a couple of years later, *Poétique des Codes sur le réseau informatique* (Editions des Archives Contemporaines, 2009). This work deploys both linguistic and cultural studies theorists (Barthes, De Certeau) in order to explain the playful acts of source code poetry, esoteric languages and net.art. While the first chapter focuses on digital literature as the result of executed code in order to develop a heuristic to approach source code, and while the third and last chapter focuses on the means of distribution of these works, particularly on the development of net.art, 1337 5p43k and codeworks, the second chapter is the most relevant to the topic of a literary analysis of source code. In that chapter, Paloque-Berges provides an introduction of creative acts in source code on both a conceptual level (drawing from Hayles and Montfort) and on a technical level. She looks at specific programming patterns and practices ("hello world", quines), technical syntax (e.g. `$`, `@` as perl tokens for expressing singular or plurals) and cultural paradigms (De Certeau's tactics vs. strategies). The second chapter of the book, in which she establishes these analyses is an example that I would like to build upon, developing closer readings of the formal, linguistic aspects of those source code pieces, going beyond strictly Perl-based poetry. Paloque-Berges's work establishes this field as a valid field of study which invites further work to be done in this dual vein of close-reading and theoretical contextualization.

Finally, *10 PRNT CHR$(205.5+RND(1)) : GOTO 10;* (MIT Press, 2012), is a collaborative work which examines the cultural intertwinings of a single line of code, through hardware, language, syntax, outputs and themes. The whole endeavour is an example of rigorous close-reading of source code, in a clearly deductive fashion, working from the words on the screen and elaborating the context within which these words exist, in order to establish the cultural relevance of source code. While the study itself, being a close-reading of only one work, and particularly a *one-liner*, is restricted in terms of broad aesthetic statements, it does show how it is possible to talk about code not as an abstract construct but as an instantiated reality. Particularly interesting is the section dedicated to the history of the BASIC programming language, and how particular languages afford particular statements and actions better in a given historical context. Finally, and similarly *Speaking Code*, the book also integrates practice-based research, in the form of ports of the original one-liner to other languages and environments, and thus contributes.

Findings

Overall, we can identify a dynamic which starts from broad theoretical work to establish a definition of software (MacKenzie) as a socio-technical and cultural object, towards close, deliberate readings of source code in more recent work (Montfort et. al., Paloque-Berges). The specific texts of source code poetry have also been given attention, in particular in terms of socio-cultural practices. These, however, focus particularly on staples such as Perl Poetry or Obfuscated C Code and specific esolangs. Because the production in this field has both increased and diversified, it should be taken into account in any subsequent study. Paloque-Berges's work is closest when it comes to analyzing the specific forms that source code poetry takes, and in how they differ from natural languages. She does so both by paying attention to syntactical tokens that make Perl uniquely suited to this kind of endeavour, but also by providing larger theoretical frameworks. Between broad cultural definitions and close-readings, it thus seems relevant to focus my research specifically on the role of programming languages, both insofar as they provide the formal specificity of source code poetry, but also act as a *material* in terms of software production, opening up a broader field of exploration than the "traditional" perl poetry and IOCCC.

Particularly, Mikhail Bakhtin and his followers develop the concept of speech-act in their critique of Saussure (*Marxism and the Philosophy of Language*, Voloshinov V., Harvard University Press, 1973). They posit the speech-act as the social event in which the speech takes on its full meaning through the active participation of all agents involved in the communication process. As such, they overcome the limitations of both the purely formal, abstract structure of syntax and the impossible exclusive subjectivity of speech. The *speech-act*, then, could be a perspective to look at source code as the socially instantiated manifestation of historically-bound formal structures. A piece of source code could be understood as a specific instance in broader intertwinings of socio-technical requirements, depending both on formal designs (the language documentation), social practices (styles, paradigms, patterns) and individual preferences (comments, idiosyncratic variable naming). The hypothesis here would be that programming languages act as both tools and resources and enable through their structures textual games that make sense within a specific regime, the regime of computation (following Hayles).

A second approach, building upon this examination of programming languages as material, would be to examine the mental constructions that are favored by programming languages (e.g. functional programming, object-oriented programming, model-view-controller, microservices, etc.). Building on works by Bogost's procedural rhetoric and Wardrip-Fruin's expressive processing, I hypothesize that it is possible to extract particular *figures de style* or rhetorical devices which happen to be particularly at play in the writing and reading of source code. Furthermore, this approach could link back to Goodman's analytical system developed in *Languages of Art: An Approach to Theory of Symbols* (Hackett Publishing Company, 1976), in order to identify how source code and programming languages re-actualize concepts of connotation, denotation and references to abstract classes and labels through an aesthetic (formal) experience. The intent is then to use the lens of programming languages in order to: 1. highlight the specificity of conceptual structures and how they relate to the socio-economic world that is being addressed in source code poetry, 2. how the syntax of these languages contribute to the specific *regime of computation*, 3. offer a perspective on how languages themselves can become textual objects through a process of appropriation inside and outside of *"standard industry practices"*.

Corpus

With this approach in mind, I've constituted a first draft for a corpus which encompasses both programs, languages, and meta-texts pertaining to programming language ecosystems, representing the evolution of the field since the publication of the last monographs focusing on the topic. A work-in-progress list of the corpus is listed in `corpus_$DATE.pdf`.

- Programming languages

Programming languages gathered include both esoteric languages in the classical sense of the term (e.g. `Brainfuck`, `Piet`, etc.) but also "non-traditional" programming languages, such as `(Qlb)`, `TrumpScript`, `UrduScript` and `Linotte`, all addressing issues of languages, locality, political systems and reading practices. While not esoteric languages as defined and studied by Montfort and Mateas, they help to shed an additional light on linguistic and dialectical practices in software. On top of those two existing categories, "traditional" languages such as `Java`, `C++`, `Lisp`, `Perl`, `Python` and `JavaScript`, amongst others are also included.

- Source code pieces

While the source code poems section of the corpus will include the “traditional” source code poetry included in previous studies (e.g. perl poetry and obfuscated code) in order to perform further close-readings on them, they will also take into account the evolution of the production in the last ten years, including the editions of the *code poetry slam*, and the *source code poetry* contests, along with isolated examples found on GitHub and personal websites. A significant addition to that corpus is the publication in 2012 of *code { poems }*, edited by Ishac Bertran, the first print version of source code poetry. This particular collection appears to be a productive starting point to develop a renewed analysis of source code.

- Peripheral texts

These additional texts include man pages (for the `ic` binary in older versions of UNIX), the `perl::bleach` module, the white paper for the `chicken` programming language, published source code books in amateur communities, code critiques from forums, writeups on code poetry contests and individual blog posts on the relationship between poetry and code and books published which deal directly with source code, ranging from *BASIC Computer Games* (ed. David H. Ahl, 1978) to *If Hemingway wrote JavaScript* (No Starch Press, 2014). I believe they help to qualify the production and transformation process that code undergoes, through additional versions, comments, modules, libraries and readings and broadcastings, requalifying it as a mutable object through inter-textuality.

These three areas -programming languages, source code, and meta-texts- will provide grounds from which I intend to investigate the multi-faceted relationship of programming languages with its instantiations in source code and the discursive contexts within which they exist and evolve, as well as the discursive contexts they create. Close readings of source code, along with esoteric and “non-traditional” languages will provide a different perspective on the formalism of programming languages, which will then be contextualized, through the meta-texts, within broader social, economic, cultural and political contexts.

Finally, following the direction of some previous research (*10 Print*, *Speaking Code*, along with Serge Bouchardon’s *La Valeur Heuristique de la Littérature Numérique*), I intend to produce additional primary sources myself, in order to test out some of the theories outlined through the examination of existing texts. These additional primary sources would include both source code written in existing programming languages (as published in *Coroutines*, Officialfan.club press, 2018) as well as the design and implementation of a new programming language.

Next steps

Based on this preliminary research and a first pass on the constitution of the corpus, I intend to direct my research on the *mode(s) of representation* of programming languages, using source code poems and other non-functional (but still meaningful) pieces of code, by alternating analyses at the level of the text (broad organization of the source code) and at the level of the line (specific reading of syntax). These modes of representation, I assume, will not be separable from the social context in which they saw the light of day, and will therefore necessarily include an inquiry into the modes of creation, distribution and maintenance of programming languages.

At this point, I envision the next steps of this research to complete my readings on specifically software as text (with Annette Vee's *Coding Literacy* and Chun's *Programmed Visions* and Berry's *The Philosophy of Software*).

From there on, there are two possibilities, either focusing on programming textbooks and manuals (e.g. *The Art of Computer Programming*, *Thinking in C++*, *Software and Mind*, *Concepts of Programming Languages*) in order to approach the process of designing languages, or continuing to read about literary criticism, linguistics and aesthetics (particularly Ong's *Orality and Textuality*, Ranciere's *Mute Speech* and *Aisthesis*, along with Burke's *Language As Symbolic Action*) in order to inform the close-reading of the gathered poems in the corpus. I would tend towards the second possibility, with the risk of remaining within a conceptual framework for longer and being only confronted later to the realities and practicalities of software/programming language development.