two - defining further the object of study

The current development of the software studies field, and particularly of the critical code studies sub-field opens up an opportunity for this research to build upon existing examinations of source code. Since the existence of beautiful source code has been recognized, and while techniques of close-reading are being elaborated by scholars of literary and media studies, there is a possibility to further explore the nature and role of aesthetics and poetics within source code.

However, I would focus specifically on *aesthetics*, rather than on *poetics*. As literary theorist Gerard Genette approaches the concept of literature through both *diction* and *fiction*, these are two complementary, yet separate approaches to the literary artistic. The difference I would like to draw here is along the lines of the different scales of representation, and between formal representation, encompassing the concrete signs which make up a work, and conceptual representation, the over-arching structures along which a work is composed and then communicated. Following the definition of Jacques Rancière, I understand aesthetics as a physical manifestation which can be grasped by the senses. Aesthetics belong to the *sensible*, whether "the movement of a light, the brush a fabric, the splash of a color" (Ranciere, 2019). These sensible moments, through their deliberate laying out by the (potential) author(s), act as symbols by virtue of being received by an audience. *Aesthetics* as delimited here are the myriad of symbols, linguistic or non-linguistic, which facilitate the communication of a given message of a given nature (e.g. a statement, an emotion, a question, etc.).

As the counterpoint to Genette's *diction*, *fiction* operates on a higher level, and is an organizing force which shapes the work as a whole into a certain configuration, and thus affords the audience of the work the possibility it to apprehend from a particular angle. Fiction here is understood as a proposal for a conceptual representation, a certain way of framing, akin to the difference between comedy and tragedy, to take the Ancient Greek example. Poetics, then, moves toward *what* is being said, while aesthetics skews towards about *how* it is being said. In that sense, it is the process of representing the world, the process of world-making, as Goodman presents it; as a global act of making, poetics operate on a macro-level, and apprehend the subject of the work (be it itself, or something external to it) in a totality. It is an operating process upon that subject, one which transforms its manifestation holistically. If poetics is operating, then aesthetics is being, an existing feature of a work waiting to be received, interpreted through active engagement. In the light of this distinction, it is those specific formal aspects, these aesthetics, that I intend to examine in source code, while acknowledging that it is impossible, and perhaps even counter-productive, to completely ignore the poetic part of source code.

So what, indeed, is the role of aesthetics within those specific texts, those programmed texts that have not been considered artworks, and will perhaps never enter a certain field of art production and reception (i.e. galleries, museums, auction houses)? The hypothesis from which I will be starting is that aesthetic features have the responsibility to communicate, and to *facilitate understanding*. A brief overview of existing literature and discourses around the beauty of code always seem to point back to a necessity to be understood —elegance, clarity, simplicity, usability are all values which seem to be central to the definitions of beautiful code by its readers and writers.

The concept of understanding as used here deserves a little more explicit attention, particularly as it exists in source code, stretched between human understanding and machine understanding. As a phenomena happening between a source and a reception, human understanding implies a general grasp of the causes, consequences of a particular state of things, and the ability to (1) act upon it immediately and (2) transpose this grasp, or the broad patterns identified, to a situation at different places, moments, and/or with different actors. Conversely, computer understanding focuses more narrowly on the ability to interpret messages and turn them into actions (e.g. the computer understands the code if it does what the programmer wanted the computer to do, through the process of transcribing that want into code).

And yet, while humans might be under the impression that they understanding something, communicating this understanding proves to be somewhat complicated, an issue that is at the core of the field of linguistics. This understanding is wrapped in codes, contexts and customs, some implicit, and some explicit. Communication between humans therefore somehow seems to always elude complete understanding. Computers, on the opposite, understanding very clearly what is being communicated to them through the medium of programming languages, and will effectively act upon it. Whether the result of that action matches the expectation of the programmer or not, a message has still been understood. It is when the computing machine has to expand upon that understanding in order to apply it to a different situation, that the process breaks down. A new message has to be communicated (i.e. a program rewritten) in order to create a new understanding. Understanding in source code is therefore stretched between this wide, general and loose appreciation of the human mind and narrow, specific, highly-efficient functioning of the computing machine. I postulate that aesthetics in source code is a set of literary features through which is negotiated this dual nature of understanding between humans and machines, and that a closer examination of these will shed a new light on the functioning of aesthetics as both a cognitive and artistic device.

In order to pursue this work, I intend to proceed as follows. After having completed a literature review on code studies, and gathering a corpus of source code which is both considered "technical" (e.g. the Linux kernel) and "artistic" (e.g. the `{code poems}` anthology), I now intend to identify the principal components of the discourses taking place around the beauty of source code, through technical textbooks, online comments, prologues to published works and existing critical literature. These components (which will feature elegance, simplicity, cleanliness, clarity, among others) will then need to be further defined in order to compose a set of aesthetic standards, a framework through which the beauty of source code can be evaluated. At this point, there will be a necessary detour by taking into account the specificity of the languages in which a given text is written. If fiction can be seen as relatively language-independent, diction, the minuteness of a comma, of a bracket or of an upper-case is, to a large extent, enforced by the language in which the program is being written, and might therefore reflect some aesthetic standards differently than in other programming languages. Finally, having constructed a framework through which the beauty of source code can be methodically approached, I intend to apply this framework to existing source code texts to see how such an interpretation might fare, and how productive such an approach might be to understand texts (in both programming and, perhaps, human languages) better. A revised version of this aesthetic framework will then be elaborated on the basis of these case studies.

A tentative outline of the work could be as such:

1. The stakes of source code as text

- Source code as an object of study (literature review)
- The reading and writing of source code (the practice of programmers -what and how do they read and write?)
- The delimitation of the corpus (from functional, to hobbyist, to artistic source code)

2. The aesthetics of source code

- The discourses of beauty around source code
- Clarity in human-machine communication
- Clarity as an aesthetic concept and its manifestations in literary theory
- A set of aesthetic features in source code

3. The linguistic materiality of source code

- The programming language object
- The programming language ecosystems
- The impact of programming languages on the set of aesthetic features in source code

4. An application to existing texts

- Case study A
- Case study B
- Case study C
- Conclusion and revision

---

steps: 1. identify the approach (lit review) 2. identify the discourses and frameworks which qualify code as beautiful 3. extract and sum up the features (elegance, simplicity, editability) 4. how does it reconnect to the aesthetic? 5. see how they are differently implemented in different languages

Machine understanding / clarity / cleanliness / elegance / simplicity

what are the aesthetic properties of source code and how do they vary based on languages?

what is the difference between aesthetics and poetics? (micro vs. macro?) —- **coming into being**

instead of finding the broader poetics, i will focus on the specific aesthetics: the creation, manifestation and appreciation of small scale formal, sensitive revelations.

poetics is activity, aesthetics is opportunity

the challenge of an aesthetic work is to figure its identity (drucker) The activity has ontological dimensions—what is and makes a work an aesthetic work—and epistemological ones—how do we know or sense (to pose this phenomenologically) that we are in the presence of an aesthetic object or experience? How, I asked myself, how does aesthetic work come to figure against the jealous ground of noise culture? And be perceived? Identified? Given place and value? (still drucker)

one of the key points is going to examine the role of *understanding.* understanding as a means to: - apprehend something: a situation (object, context, possibilities of actions, consequences of actions) - refer to something else at a different time and place, while still being able to draw a parallel

is there a difference between human understanding and machine understanding? -> perhaps human understanding is unbounded by time and place, while machine understanding has to be clearly delimited.

on the opposite side, however, human understanding is fuzzy at best. it doesn't quite know exactly what it is dealing with (wittgenstein: all philosophical problems are formulation problems). in that sense, the machine is already extremely clear, but within a very constrained set (e.g. chess)

the machine understands chess insofar as it can play chess. it doesn't understand chess (so far) such that it could apply the patterns of chess to the patterns of some other rule domain. human understanding of chess exists without the chessboard.

how do we make those two understanding coexist, then? it goes through language: language is key in both narrow (math) and broad (poetry) definitions. the laying out of language (formulation and writing) then allows (to some extent) for those understandings. the broad conceptual maps (**poetics**) are actually language-independent. a plot twist is a plot twist in any language. However, the way you reveal the plot twist could actually vary from language to language.

aesthetics enable the understanding (goodman), while poetics provide a framework which *tilts*, or influences the approach of that understanding. it is not what is meant to be understood, but it is *the presented facet* of what is supposed to be understood. (e.g. qualitative vs. quantitative, or drama vs. comedy)