

Universiteit Leiden

Master Creative Intelligence & Technology

A Poetics of Code

Name:	Joseph Catlett
Student ID:	4229746
Date:	31/10/2026
1st supervisor:	Edwin van der Heide
2nd supervisor:	Max van Duijn

Master's Thesis in Creative Intelligence & Technology

Leiden Institute of Advanced Computer Science
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

A Poetics of Code

Joseph Catlett

Supervised by Edwin van der Heide & Max van Duijn

October 31, 2026

```
    static signed char a[] = {0x69,
  110, 118, 97, 108, 105,  0x64, 1-1,
0x6d, 111, 118,  101, 1<<1<<1<<1<<1<<1,
114,  105, 0x6e,  103,  32, 'o'/3, 100,
32, 102, 114, 111,  0x6d, 32, 115, 116,
97, 100-001, 107, 32,37, 2*'2', '@'>>1,
116, '%' + '%' + '%','w'-'W',115, 0x74,
97, 3*'!', 107, 'q' - 'Q', 37, 10*'\n',
10, 0}, * b = a + (1<<1<<1<<1), * w, x,
*q, c, r; int main(int d, char *e []) {
return q = (signed char *)(e+1+1), (r =
e[0] && e[1] ? 0 : 0 * puts (a) + 1) ||
(r = e[1<<1] && d != 1 <<1 && 0 * puts(
a) + 1) || e[1- -1] ||  (r = atoi(e[1])
< -0200 || atoi (e [1]) > 0x7f || ( x =
atoi( e[1] ) ) == 0 ? 0 * puts(a) + 1 :
0) || e [1- -1] || (x- -x > 1-1 ? (q[0]
  = x, q[1] = q[3] = 1, q[2] = 2) : (
    memset ( w = ( signed  char * )
```

```
    malloc(-x), '1', -x), puts (w),
  q[0] = x, q[1] = '0', q[2] = q[3] =
0)), r || (q[3] ? (c = 6 - q[1] - q[2],
(q[0] != 1) ? q[0]-- , d = q[2], q[2] =
c, main(2, e), c = q[2], q[2] = d, q[0]
++ : 0, printf(b, q[0], q[1], q[2]), (q
[0] != 1) ? q[0]--, d = q[1],q[1] = c ,
main(2, e), c = q[1], q[1] = d, q[0] ++
: 0) : - 1 - q[0] - 1 == 0 ? (w[- x - 1
- (q[1] & 1 ^ 1)] = q[1], puts (w), w [
- x - 1 - (q[1] & 1)] = q[1], puts(w) )
: - 1 - q[0] == 0 ?  (w[- x - 1] = q[ 1
  ], puts(w)) : (q[0] += 1 + ( q[1] & 1
       ^ 1), main(2, e), q[0] -= 1 + ( q
             [1] & 1 ^ 1), q[1] & 1 ? (q
                [0]+=1+1,  q[1]^=1, main
                (2, e), q[1]^=1, q[0]-=1
                +1) : 0, w[q[0] - x] = q
              [1], puts(w), q[1] & 1 ?
           0 : (q[0]+=1+1, q[1]^=1,
         main (2, e), q[1]^=1, q
        [0]-=1+1), q[0] += 1 +
     (q[1] & 1),main(2,e)
 , q[0] -= 1 + (q[1]
& 1) ) ), r; }
```

Abstract

Software and code are conventionally approached through the mathematical and computational frameworks of logic, function, and execution. Yet source code is also human-produced, shaped by linguistic choices, cultural conventions, and the situated perspectives of those who write and read it. This thesis investigates the intersections between programming, linguistics, and poetics, asking whether collective close reading can produce an epistemic shift from teleological interpretation toward poetic modes of understanding. Two multidisciplinary groups participated in three collective close-reading sessions of contrasting code texts. Their discussions, interpretive tensions, disciplinary hierarchies, and speculative readings were transcribed and qualitatively analysed. The findings suggest that collective close reading can facilitate poetic engagement with code, although unevenly and in ways strongly conditioned by social and disciplinary dynamics. Technical interpretation did not disappear; instead, function, efficiency, structure, naming, repetition, and convention often became material for poetic, affective, linguistic, and political interpretation. The study therefore proposes a poetics of code in which technical knowledge becomes one mode of knowing among others.

Acknowledgements

I want to extend my sincerest thanks to all the participants who dedicated their time, attention, and openness to this research. I am also deeply grateful to Tamara de Groot, whose guidance through personal correspondence helped shape the experimental design and my approach to facilitating the collective close-reading sessions.

Finally, I want to also offer my deepest thank you to Yusser al Obaidi, who was not only an inspiration for this research but whose work and personal correspondence have been actively shaping and developing this framework of poetics both prior to and throughout the work. From her appearance on *Off-Margin w/ Amy Gowen* by Radio WORM, to her seminal bachelor's thesis, *Dear Barzakhiyaat*, to her essays published in *Promiscuous Infrastructures: Practicing Care*, her investigations into Sufi epistemology, the connections she draws to Aimé Césaire, and her discussions of work from authors, philosophers, and activists such as Audre Lorde and bell hooks, have profoundly informed my own relationship with poetics and my attempts to "free poetics from the discipline of poetry. To claim poetics without claiming poet." (Gowen and al Obaidi, 2024, 24:10)

Contents

1 Introduction **6**

2 Background **7**
2.1 Pre-existing Linguistic Qualities in Code . . . 7
2.2 Poetics . . . 12
2.3 Close Reading . . . 26

3 Research Question **29**

4 Methodology **33**
4.1 Structure . . . 33
4.2 Content . . . 36
4.3 Analysis . . . 38
4.4 Addressing Research Question . . . 40

5 Results & Analysis **41**
5.1 Social Conditions . . . 43
5.2 Interpretive Authority . . . 47
5.3 Speculative and Non-teleological Discussion . . . 55
5.4 Poetic Teleology . . . 63
5.5 Poetic Categories and Features . . . 67
5.6 Epistemic Shifts . . . 89

6 Discussion **93**
6.1 Assessing Indicators . . . 93
6.2 Research Questions . . . 98

7 Conclusion **100**

A Code Selection: Adder **108**

B Code Selection: Website Evidence Collector > Inspector **109**

1 Introduction

The production of software has been firmly relegated to the domain of computer science and engineering. Sprouting from mathematical fields such as boolean algebra, logic, and calculus, it is seen as an intrinsically (and exclusively) arithmetic discipline. But how true is this presumption?

While coding indeed demands intimate knowledge of these mathematical subjects, there is still a very *human* engineer behind the keyboard. That human engineer makes decisions; decisions emergent from an individuated, subjective experience, ones that reveal their perspective on what is an elegant, effective, and efficient solution. Many of these coding problems have multiple suitable strategies that make discriminating between them seem more like a creative liberty than the mechanistic executions of some deterministic decision tree.

When software engineers write their code, they do so also with intent to communicate; they communicate with the machine, with future reviewers, colleagues, students, and any other who accesses their repository. Their train of thought, their goal, their solution, are all encoded in the programme produced. Is it possible then, that through this, the code produced captures some essence or character of the author? That the code written by the author is actually some expression of themselves or broader software culture in that moment?

These questions have led us to hypothesise that code may have a poetic dimension. That embedded within it is the subjectivity of the author. Much like how we do not see architecture only as an act of building formal shelters, but also as a form of sculptural art, software may not be only an act of employing algorithmic thinking, but also a form of producing poetic text.

My allusion to the world of architecture is not trivial. Architecture is an example of a creative paradigm with strict formal traditions, and yet, still has designers who remain capable of producing structures which are artistic and expressive. There are aesthetic motifs that exist within architecture that can be leveraged to communicate ideas from the architect. So it is not the absence of formal technique or rigid structure that lends itself to poetic expression. For code, a medium that is defined by its structure, by its rigidity and formalism, architecture serves as an example that one can be expressive within such conventions and constraints. Furthermore, we employ architectural examples as a reference to one major inspiration for this line of contemplation: Gaston Bachelard's "The Poetics of Space", wherein Bachelard interprets the forms, structures, and imposing associations of architectural features as poetic in quality. To Bachelard, what at first seems to be simple, geometrical objects, can actually be reinterpreted, reframed, and reseen as something deeper in meaning. Bachelard asserts that it is in our relationship with architecture that it becomes poetic, "A house that has been inhabited[1] is not an inert box. Inhabited space transcends geometrical space" (Bachelard, 1969, p. 47). Following this investigative thread, we

[1]Some translations use the word 'experienced'. The original French is written "La maison vécue n'est, pas une boite inerte." (Bachelard, 1961, p. 73)

approach the central theme of this research: can code too be poetic?

2 Background

When initiating this investigation into the poetic dimensions of code texts, we must first understand the landscape upon which the discussion meanders. How are poetic dimensions investigated and explored? What do we even mean by poetic dimensions and poetics? Are there preconditions to being poetic, and if so what are they? We will answer these questions, beginning with an investigation into the linguistic dimensions of code. We will then construct a poetic scope to be employed throughout our investigation. And, finally, explain the method to be implemented in our research.

2.1 Pre-existing Linguistic Qualities in Code

Gaston Bachelard studied the segments of a house as those of a poem. He ruminated and fantasised about the metaphorical quality of roundness and corners, attics and cellars, the minuscule and massive. Bachelard does not attempt to propose any justification for the appropriateness of poetic interpretation of architecture beyond the poetic quality of phenomenological experience itself[2].

However, in the case of code texts, we find it important to consider the textual (and, perhaps, therefore linguistic) qualities of code. Code, in its textual format, becomes a textual medium. It demands an author's engagement with language and semiotics, creating a much stronger connection to poetics through the transitive relation to language. This section is a short, but necessary detour, to understand the linguistic foundation, upon which code rests; the very same foundation found supporting the structures that lead to the writing of 'traditional' poems.

2.1.1 Semiotic Paradigms

Language exists as a semiotic system; a network of signs and symbols giving structure to concepts and a means to communicate effectively about the internal landscape of experience. Semiotics, thus, forms an important underlying component to language. Much linguistic, philosophical, and neurophysiological research has been invested into understanding the semiotics of language. Following from this, recognising the similarities between linguistic and code semiotics is a significant first step to generating our techno-poetic image of code.

Kumiko Tanaka-Ishii (2010) explains in her book, *Semiotics of Programming*, that programming languages offer a unique opportunity for linguistic analysis seeing as these "expressions written in programming languages are interpreted by both humans and machines" and as such "reflect the lin-

[2]This idea of the legitimised authority of phenomenological experience is not an idea we reject. We discuss this in more depth in the later section on defining poetics.

guistic behaviours of both"[3] (Tanaka-Ishii, 2010). Tanaka-Ishii (2010) goes on to compare functional programming languages and object-oriented programming languages to Saussure's dyadic[4] and Peirce's triadic[5] semiotic model, respectively (Tanaka-Ishii, 2010).

> "Because all dyadic identifiers consist of a name and its content, the name is likely to correspond to the signifier and the content to the signified. [...] Through the name *Rectangle*, the set of features (i.e. the *width* and *height*) is collected. Such correspondence of the signifier and the signified also holds for identifiers that stand for functions: the function name *area* is a signifier representing a procedure of mapping as the signified.
>
> [...]
>
> A sign in the triadic model has representamen, an object, and an interpretant. Since all triadic identifiers in the object-oriented paradigm consist of a name, data, and functionalities, these lend themselves respectively to the comparison with the relata of the triadic sign model." (Tanaka-Ishii, 2010, pp. 36-37)

On a semiotic level, we can map the structures of programming languages to the conceptual structures of linguistic engagement in humans. How information is objectified, conceptualised, and represented in code, is therefore very similar to how it is in human speech and literature.

Tanaka-Ishii demonstrates that the same emergent patterns that underlie the vast and complex structure of our language are also present in the foundations of programming languages. They share the same skeletal anatomy that gives them their structure. With these conceptual frameworks being shared, we are prompted to recognise parallels on the most fundamental level: how a language deals with the relationship between signs and significations. Now, with Tanaka-Ishii having shown that our semiotic frameworks are applicable, we can re-build the frameworks that describe language up to the highest levels of abstraction that facilitates its application to poetry.

2.1.2 Linguistic Power Laws

Popular linguistic features such as distribution and growth of vocabulary have been demonstrated to exist in similar fashion in both kinds of lan-

[3] Tanaka-Ishii does not directly define the *linguistic behaviour of machines*, however, in the surrounding paragraphs she does describe certain characteristics of artificial machine languages, such as their demand for "mere mechanical command chains", their fundamental nature as "chains of substitutions", or their reliance on "open and explicit" notation systems (Tanaka-Ishii, 2010).

[4] Saussure's dyadic semiotic model asserts that all *signs* are composed of two parts; the *signifier* (the form, sound, or representation) and the *signified* (the concept being substituted by a representation) (Chandler, 2006).

[5] Peirce's triadic model adds a third actor to the system. We have an *object* (the thing being represented and interpreted), the *representamen* (the form or substitution representing the object), and the *interpretant* (the sense made of the sign in the internal mind of the observer, which may indeed be another sign) (Dahlstrom and Somayaji, 2003).

guages, programming and human. Zhang (2009) demonstrates that languages such as Java and C++, throughout 10 years of analysed development, follow Zipf's Law[6], the adapted form Zipf-Mandelbrot Law[7], and Heap's Law[8] (Zhang, 2009). Many sources also corroborate the Zipf-Mandelbrot power law property across different programming languages (Casalnuovo et al., 2019; Kokol and Kokol, 1996; Louridas et al., 2008; Pierret and Poshyvanyk, 2009).

Kokol and Kokol (1996) concluded that linguistic metrics could form the foundation of software analysis. Using linguistic metrics, such as Zipf's Law, they found that adherence to them was a predictor of the *programme mark*[9] of a coding language (Kokol and Kokol, 1996). They go on to say in their conclusion, "This finding opens the possibility of using linguistic laws as a basis for developing a new class of software metrics (i.e., linguistic metrics). Our study shows that linguistic metrics can perform better in the prediction of some software quality characteristics and are viable candidates for future research in the software metrics area"(Kokol and Kokol, 1996).

What we see here is that programming languages have word distributions that follow our natural language. Linguistically, programming languages are structured and distributed in the same way as the languages we have developed for speech and literature; those same languages we use for poetry. Kokol and Kokol (1996) go even further, showing us that while a passive adherence to power laws and linguistics structures is indeed true, these actually say something about the programming language. They show us that adhering to these rules is not simply a quirk or convenience for readability, but actually predicts quality across a number of dimensions.

2.1.3 Linguistic Abstraction

From this common underlying semiotic structure and similar statistical token density are higher-level emergent properties that are also shared across the human and machinic paradigms. Visser (2015) explains how the phenomenon of *linguistic abstraction* is an important part of programming languages, comparing them to *design patterns* in software development (Visser, 2015).

A *linguistic abstraction* is the process of creating language constructs that abstract away from concrete details. In human language, we see these as terms for concepts that move away from atomic or concrete phenomena; non-material realities such as love, democracy, or justice. We do not need

[6]Zipf's Law states that the frequencies of some events, f, are proportional to their rank, r. In a linguistic context, this would mean the 2nd most common English word, appears 1/2 as often as the 1st, the 3rd appears 1/3 as often, the 4th appears 1/4, and so on.

[7]The Zipf-Mandelbrot Law is simply a more generalised version of the Zipf-Law, introducing parameters to make it more broadly applicable.

[8]Heap's Law is a prediction of vocabulary of a text corpus, stating that the number of unique words is approximately $\sqrt{N}$, where N is the total number of tokens in the corpus.

[9]The *programme mark* of a coding language is an average grade calculated from the evaluations of a language by (at least) five coding experts over the following properties; readability, robustness, user interface, programming style, and reliability (Kokol and Kokol, 1996).

to understand all the chemo-physiology mechanisms of the brain nor have an exhaustive understanding of human-to-human social evolution and its materialisation in modern relationships to understand what is meant when someone says to us, *"I love you."* That is linguistic abstraction.

This allows for more general and efficient expression of concepts. It involves moving from specific instances to broader categories or representations, often using functions, naming, and parametrization to encapsulate complex operations (Nørmark, 2013). Visser (2015) describes the process occurring in software engineering:

> "A linguistic abstraction such as procedural abstraction *captures our understanding of a concept in software*. Over time the understanding of the abstraction in terms of the original implementation model erodes. New programmers learn to programme with procedures without ever learning the underlying implementation scheme. The concept is no longer a convenience, but a first-class concept in thinking about software construction." (Visser, 2015, p. 12, emphasis in original.)

Visser (2015) goes on to provide a number of examples of this process of design pattern to linguistic abstraction and claims that this process of evolution from explicit design patterns to generalised linguistic abstractions is essential to many areas of software engineering (Visser, 2015). What we are seeing is that people employing programming languages are doing so in ways similar to the employment of natural language. We also see that the reaction language has to its usage, the forms that emerge, the changes in complexity, are mirrored across the two linguistic domains.

2.1.4 Metaphors & Metonyms

With the richness of abstraction, coding languages become a potential site for poetic phenomena such as metaphors and metonyms. However, these linguistic parallelisms rarely have their analogous linguistic forms acknowledged. Exman and Plebe (2016) directly compare coding structures to structures found in prose and poetry. They map poetic features to software terminology such as 'running', 'classes', 'refactoring', among others (Exman and Plebe, 2016). Exman and Plebe (2016) go beyond the examples we have seen until now, which are solely to demonstrate a linguistic relation, and attempt to now draw parallels on a higher level of abstraction, one that is poetic.

> "Repeating a poem to myself (mentally or aloud) corresponds somehow to "executing" it. That's interesting. As the execution of software will affect hardware components, registers, memory content and so on, the "execution" of a poem will elicit responses, in emotional brain centers, recall long-term memories, activate semantic networks. In both cases the meaning of the code (poetic or software) is in the activation resulting from its execution." (Exman and Plebe, 2016, pp. 9-10)

However, what separates this from any other form of writing, say a novel, newspaper article, or even a formulaic recipe? Exman and Plebe (2016) assert that it is in the paradoxical relationship between structural constraint but grammatical freedom, which is likened to object-oriented programming's more formal structure but more freedom in 'conceptual terms' (Exman and Plebe, 2016). These similarities inspire a new software design pattern, mimicking poetic metaphor, demonstrating the conceptual potential of allowing cross-pollination between these two fields and writing paradigms.

2.1.5 Language & Mathematical Aptitude

We have now seen a number of examples of similarities between these forms of language, both in terms of structure and form (see *Linguistic Power Laws* and *Semiotics Paradigms*) and also in application and development (see *Linguistic Abstraction* and *Metaphors & Metonyms*). However, the similarities also extend to how we learn language. Gordon (2024) points out that it has already been established (and for even longer been suspected) that an aptitude for learning natural languages is a predictor for an aptitude toward learning to code (Gordon, 2024). Gordon (2024) references Prat et al. (2020) in his argument, wherein they show literacy and natural language aptitude accounts for 17% of the variance in individual aptitude for Python, while numeracy accounts for only 2% (Prat et al., 2020). To reiterate, there is an eight-fold increase in the ability of linguistic aptitude to predict your ability to learn a programming language over mathematical aptitude. This statement alone disrupts the cultural associations we have of what we need to know and learn in order to understand (how to) code. Their research demonstrates how the emphasis on numeracy and mathematical aptitude in software pedagogy may be misplaced and strongly suggests that educational environments instructing programming may benefit from taking inspiration from existing linguistic literature (Gordon, 2024; Prat et al., 2020).

This idea of modelling programming language pedagogy after natural language education has a historical basis. Computer programming languages were designed to parallel natural grammatical structures, something first formalised by the linguist Noam Chomsky, over half a century ago (Chomsky, 1956).

2.1.6 Authorship & Subjectivity

Another example of the human relationship with language is in authorship. In natural language analysis, machine learning is often used to identify the author of a text among a selection of candidates. Bayrami and Rice (2021) discuss their application of similar machine learning techniques to classify authorship of code text, focusing on features that capture writing style (Bayrami and Rice, 2021). Features taken from the source code along with sociological features of authors are combined to develop a classification model in C++ that was able to identify an author's identity with an accuracy of 75% (Bayrami and Rice, 2021).

This suggests a certain level of human subjectivity in code. Similar to

natural language, where a text can capture the essence or quality of the author, so too can code text, dispelling the myth of mathematical detachment or mathematical objectivity; code is expressive in some capacity.

Anecdotally, this is something many software engineers and computer scientists may experience when trying to read through the code bases of peers or strangers. There is an element of trying to understand the author's intentions or thought-process rather than simply engaging with the code alone. Understanding the positionality and intentionality of the author is a helpful factor in interpreting a code text.

2.2 Poetics

When designing an experiment that investigates the *poetics* of a given phenomenon, whether it be conventional or unconventional, as is the case with code texts, it is important to be clear with what we mean by "*poetics*". We will discuss in detail the origins of our working definition of poetics and also contrast it with other popular (mis)understandings of what poetics is. First and foremost, what we are not attempting to do is convince a group of people, or you, reader, that code texts and poems are equivalent. Furthermore, it is not the aim of the methodology to 'convince' participants of the poetic nature of code. Rather, the goal is to open a space where such a relationship might emerge and be observed.

We understand poetics more as an epistemological framework. Poetics is a way in which one can engage with, understand, and learn about artefacts and objects; in a sense, it is a verb (al Obaidi, 2024, pp. 120–121). When we read a poem, we apply poetics in that we understand the literal words on the page and their immediate semantic definitions do not encompass everything the author was trying to communicate. Bachelard does this to architecture in "The Poetics of Space" wherein he does not seek to collapse meaning into designer's intention, but instead as a meandering of the phenomenological experience of the dweller. However, we can go further and also posit that poetics is not about examining the words and their subtext to understand *only* the author's true intentions for communication; poetics also is a liberating practice of interpretive authority, de-monopolising meaning, taking it from the author's hands, unfolding and magnifying it, and redistributing it to the others who engage with the interpretive artefact.

This idea is adjacent to that discussed in Roland Barthes' famous essay, "Death of the Author". Barthes (1967) explains that from a linguistic perspective, an author is nothing except the one who happened to do the writing in the same way that *"I"* is nothing more than the person who says *"I"*; language knows only subjects, not persons (Barthes, 1967, p. 51). The author of a text (or the architect of a building, or the programmer of a piece of code) has no more control over the way in which that text will persist in the world than the symphony of cultures that informed their perspective anterior to creation, nor the minds that interpret it after they have released it from beneath their pen. Meaning, then, is produced by a dialogue of author-object-reader. And, for Barthes, it is that final component that has

the ultimate say:

> "Here we discern the total being of writing: a text consists of multiple writings, proceeding from several cultures and entering into dialogue, into parody, into contestation; but there is a site where this multiplicity is collected, and this site is not the author, as has hitherto been claimed, but the reader: the reader is the very space in which are inscribed, without any of them being lost, all citations out of which a writing is made:" (Barthes, 1967, p. 54)

We depart from here, a site where we acknowledge the power of the subjective experience and interpretive authority of *the reader* (though "*the reader*", here, is simply a pseudonym for the one who experiences that which is to be interpreted poetically). We seek to develop a working definition of poetics through distillation and re-imagination of thinkers and poets like Gaston Bachelard, Aimé Césaire, and those from Sufi poetic schools of Taḥqīq such as Muhammed ʻAlī al-Tahanāwī and Mīrzā ʻAbd al-Qādir Bīdil. We will also position ourselves in contrast to conventional definitions, like those that derived from Aristotelian poetics, to illustrate the differences and elucidate the specificities of our position.

2.2.1 Aristotle

One of the most influential pieces within contemporary Western European conceptualisations of poetics is Aristotle's *Treatise of Poetics*. Aristotle decomposes poetics into a network of dimensions with which one has the ability to produce artistic *mimesis* (Aristotle, 2008). Poetry, for Aristotle (2008), is inherently mimetic, in that it is an attempt by someone to imitate the world or to extract from the world some truer form. He elucidates this by contrasting Homer and Empedocles, explaining that both use verse and metre in their texts, yet despite this we do not refer to Empedocles as a poet, rather a physicist[10]. Poetics was not then exclusively about linguistic features of the text, but the culmination of an author's intentions when interacting with (or facilitating an interaction between) medium, object, and manner as a means to produce an artefact for the gaze of an audience (Aristotle, 2008).

As an audience, Aristotle (2008) argues that it is this mimetic quality that creates space for artistic appreciation. Imitative accuracy and efficacy can produce a sense of recognition and satisfaction. He goes further, arguing imitation is a fragment of the quintessence of *being*. It is something we condition into children and instinctively practice as social, mimetic creatures (Aristotle, 2008). However, Aristotle (2008) goes on to say that without previous exposure to the original, appreciation can still be generated through

[10]Empedocles was a physicist and philosopher famous for originating the four universal elements governing worldly interactions, *earth, fire, air,* and *water*. Interestingly, this cosmological text was written in hexameter verse making it, at the very least by purely structural standards, quite poetic (Kingsley and Parry, 2024).

craft, skill, and implementation of language embellishment, the latter of which he refers to as *artistic ornament* (Aristotle, 2008). For Aristotle, poetics is metaphor meets ornament.

Many of Aristotle's ruminations of poetics permeate our definitions of poetics today, especially in a Euro-Western philosophical context. The Oxford English Dictionary (OED), defines poetics as:

> 1.a. The aspect of literary criticism that deals with poetry; the branch of knowledge that deals with the techniques of poetry. Also: a treatise on poetic art, spec. that written by Aristotle.
>
> 1.b. The creative principles informing any literary, social or cultural construction, or the theoretical study of these; a theory of form.
>
> 2. Poetic composition; the writing of poems. Obsolete. rare. (Oxford English Dictionary, 2025)

These definitions remain very strict with their inclusion of what mediums are poetic, prescribing only poetry or literature. However, it offers insightful dimensions of the *poetics* that govern *poetry*. What we have is something that can capture the explicit *creative principles* of an author's work. We can analyse their metre, verse, alliteration, simile, etc. to produce an image of their works' composition poetically. Simultaneously, this definition includes more implicit dimensions, ones that might be obscure to the reader, but also obscure to the author themself. These include the formal and informal analysis of structures, devices, and norms that enable the external environment to influence a text, such as the cultural system in which it was produced, the contemporary discourses at the time, or the accepted expectations of genre specific pieces. Poetics becomes both the principles the author subscribes to consciously and the external structures which prescribe principles upon the author.

What we witness is a divergence from hermeneutics or 'meaning making'. Poetics is less about the actual meaning of any one author's work and more about the structures, conventions, or contexts that make meaning possible (Culler, 2015). We depart from a strict comparison of Aristotelian mimetic artefact and its referent, from the myopia of ornaments and metaphors. Olsen (1976) identifies a strategy for this meaning-making formalisation which he refers to as semantic (Olsen, 1976). The central insight to semantic theory is that terms have primary and secondary meaning. Primary meaning is taken for granted and falls under the prescribed, accepted definition of a word. Secondary meaning, by contrast, is unprescribed, emergent from a term's associations to external objects, events, situations. The secondary meaning is often also referred to as a 'connotation' or 'implied meaning' (Olsen, 1976). These connotations can be overt or implicit, much like the principles governing an author's work. When an author produces a poetic text, they may be unaware of existing associations to a term, they may be drawing upon one that is idiosyncratic, or the reader may have one individual to themself. All of these produce an intersubjectivity to poetic

text. Poetics becomes more of an emergent property from the interaction between author, object, and reader.

2.2.2 Gaston Bachelard

Bachelard (1969) attempted to generate a way to poetically analyse architecture, to produce an arsenal of techniques and patterns suitable for analysis of poetic meaning in architectural form; this step of form interpretation is more hermeneutic than poetic, but as Culler (2015) mentions, "[i]n practice [...] the two enterprises blend into each other..." (Culler, 2015). Bachelard (1969) discusses this intersubjective nature between the triadic reader-writer-object network. It is the interaction between the three that produces a quality beyond the instrumental function of shelter.

Bachelard's (1969) *The Poetics of Space* proposes neither a theory of space nor a psychology of imagination, but a disciplined attentiveness to the ways in which space is *lived* through poetic images. His dissatisfaction with what he names the "growing rationalism of contemporary science" does not result in an opposition between imagination and knowledge but instead leads him to question modes of knowing that suppress imagination in the name of explanatory clarity (Bachelard, 1969, p. 1). In the foreword to the 2014 publication, Mark Z. Danielewski captures this sentiment excellently describing Bachelard's "distaste [...] for what impedes in the name of dogma. He values the imagination because he recognises that understanding without imagination is doctrine without growth" (Danielewski, 2014, p. xv).

Central to Bachelard's project is a refusal to treat poetic images as objects requiring reduction. He is against psychoanalytical and psychological readings that seek to explain images through definitive conclusions or ontology. The poetic image, he argues, is personally innovative. It cannot be traced back to causal origins without losing its force. This is almost an echo of Barthes' (1967) comments on the Critic, "everything is to be *disentangled*, but nothing *deciphered*, structure can be followed, *"threaded"* (as we say in a stocking) in all its reprises, all its stages, but there is no end to it, no bottom; the space of writing is to be traversed, not Peirced; writing constantly posits meaning, but always in order to evaporate it: writing seeks a systemic exemption from meaning." (Barthes, 1967, pp. 53–54, emphasis in original). To receive an image is not to decode it, but to encounter it as an event, and explore it. This encounter is marked by what he calls reverberation; an affective resonance that precedes interpretation and situates the reader within a shared, inter-subjective, inter-objective field.

Imagination, in this context, is not a faculty of materialisation opposed to reality. It is a mode of attention. "We must be fully attentive to the image at the very moment it appears, both as itself and as a vibration of the psyche. [...] the notion of attention was key" (Kearney, 2014, p. xix). Bachelard (1969) describes imagination as both receptive and creative, an "acoustic listening" that participates in what it perceives (Kearney, 2014, p. xx). The listener becomes situated, so too do the acoustics, as well as what is emergent from the interaction between the two. What is derived is an entanglement of the participant-object-creator triad. The poetic image does

not pass from subject to object or from object to subject in a linear way; it establishes a circuit in which passivity and activity become inseparable (Kearney, 2014, p. xx).

The emphasis on inhabitation becomes explicit in Bachelard's treatment of space. His recurring figures (the house, the attic, the cellar, the corner) are not symbolic codes to be deciphered, but sites of *lived tension*. The house is vertically polarized; the attic and cellar mark opposing but complementary orientations of experience, clarity and obscurity, elevation and depth (Bachelard, 1969, pp. 38–39). These are not metaphors imposed upon space, they are ways in which space becomes meaningful through habitation. Through situated experience. The dialectics of inside and outside, immensity and miniature, are not resolved but sustained. They structure experience by holding *antinomies*[11] open.

Such antinomies are not failures of thought. They are conditions of poetic knowing. Bachelard repeatedly returns to dichotomy. Inside and outside, rationality and reverie, precision and excess. He does this to show how poetic attention moves between dichotomy and exists precisely within the tension. The "intellectualist philosopher" who seeks to stabilise meaning through precise definition is unsettled by "the poet's daring" (Bachelard, 1969, p. 165). A curve may be warm, an angle cold. A slight modifier alters the entire affective geometry of thought. Meaning is then not fixed but emergent through nuance rather than structure.

This attentiveness requires a corresponding discipline of unknowing. For Bachelard, knowing must be accompanied by an equal capacity to forget knowing (Bachelard, 1969, p. 16). This forgetting is not ignorance, but a difficult transcendence of established categories (Bachelard, 1969, p. 16). To receive a poetic image, one must suspend the impulse to *explain* it, to anchor it within a familiar conceptual framework. The image transcends the premises of sensibility precisely because it asks the reader to live what has not yet been lived, to become receptive to "an overture of language" (Bachelard, 1969, p. 17).

Bachelard's poetics can thus be understood as articulating a phenomenology of realisation. Knowledge is not secured through explanation alone, but verified through lived adequacy. To know is to inhabit; to imagine is to participate. In this shared commitment to realised, situated understanding, poetics and science no longer stand opposed. They become a dialectic, two individual and equitable modes of engaging the world through attention, resonance, and dwelling.

2.2.3 Aimé Césaire

Aimé Césaire's definition of poetics directly addresses the nature of the dichotomy between science and poetics that makes his discussions and theorisations useful for understanding poetics in this investigation. As al Obaidi (2024) puts it, "[h]e posits the poetically conjured up image as a manner to transcend the limitations of scientific knowledge." It has three positions

[11] see discussion on Aimé Césaire.

that are primarily important for understanding what poetics means in the context of this research: its perspective on tension, identity, and relation to scientific knowledge.

Césaire discusses the concept of dichotomy and antinomy throughout his explorations of poetics in *Lyric and Dramatic Poetry*. He gives his principal example in the sentence:

> "There we see Lyric and Dramatic Poetry resolved - and by the poetic state - two of the most anguishing antinomies that exist: the antinomy of one and other, the antinomy of the Self and the World." (Césaire, 1990, p. xlix)

For Césaire, antinomies are not simple contradictions. He understands them in the Kantian sense, as *necessary* opposing principles that are irreducible to one another and unsolvable through ordinary logic. The antinomies, as Césaire sees them, are structural tensions not to be formally resolved or collapsed to a singular unifying truth. In fact, they are the fundamental poetic component that shapes experience rather than blocking it through contradiction. Césaire does not argue that poetry unifies antinomies, but instead, poetics holds the contradiction, inhabits the tension, and transforms it into lived intensity. Similar to Bachelard's rejection of the intellectual philosopher consistently collapsing dichotomy and dialectics into a single stabilised interpretation.

This is seen in Césaire's example of Self and World, neither the Self nor the World can be reduced to one another (we occupy an interior experience that the outside world cannot access without our operation upon it, but the exterior world cannot be accessed without action and attention by the interior; they are distinct and in contrast, yet mutually necessary to produce an 'experience'). This tension, this relationship, is where poetics resides for Césaire. It is in the poetic state that the Self and the World co-exist without collapsing into sameness. It is in this state that the cosmic and the human interact, coalesce, and "blossom" (Césaire, 1990). Contradiction becomes expressive rather than paralysing.

This *poetic state* is an important concept for Césaire. "[T]he poetic state" is this residence in the interstice of the World and Self, but what is important to note is that it requires active process and intention to remain there. For Césaire, poetics is not a static object or formal category. Poetics is an active operation, something ongoing and happening.

> "FIRST PROPOSITION:
>
> Poetry is that process which through the world, image, myth, love and humour establishes me at the living heart of the world" (Césaire, 1990, p. lv)

The poetic state is the act of remaining in sanctimonious relations. This puts the poetic in contrast to the Aristotelian poetics and in alignment with the Bachelardian poetics, positioned as an intersubjective-interobjective relation between author, reader, and artefact. It is relational, not an identity

or formal set of features. Poetics ceases to be a list of aesthetic techniques and forms that transform text into a poem, but rather a dynamic relationship set in motion and sustained by a reader in their reorientation of themselves with the text. For Césaire, poetry is not self-expression or representation of the world, but a dynamic process that brings the poet into being at a charged, living point where two sides of an antinomy, like Self and World, encounter one another intensely. The poetic act, in this case, establishes the Self in relation to the World. Poetics becomes a theory of relation, a mode of attention and being (Césaire, 1990; al Obaidi, 2024, pp. 117–121).

What should be understood from this is not that poetic knowledge is attempting to *explain* phenomena that scientific practices are unable to. Epistemologically, poetics is a fundamentally different endeavour. It is not inferior to science, it is simply distinct. Epistemically distinct in its values, goals, and methods. Aimé Césaire would assert that scientific knowledge isolates objects, abstracts, and seeks neutrality. Césaire says,

> "Science affords a view of the world, but a summary and superficial view.
>
> Physics classifies and explains, but the essence of things eludes it. The natural sciences classify, but the *quid proprium* of things eludes them.
>
> [...] In short, scientific knowledge enumerates, measures, classifies, and kills." (Césaire, 1990, p. xlii, emphasis in original)

Poetic knowledge, with its triadic, relational structure, implicates the knower and explicitly situates and positions knowledge. Rather than reducing and isolating, poetics attempts to (re)imbue objects with a depth; to capture the complexity of the relations they have interobjectively and intersubjectively. Césaire states this in his third and sixth poetic propositions:

> "THIRD PROPOSITION:
>
> Poetic knowledge is characterised by humankind splattering the object with all its mobilised richness.
>
> [...]
>
> SIXTH PROPOSITION:
>
> Scientific truth has its sign as coherence and effecity. Poetic truth has as its sign beauty." (Césaire, 1990, p. lv)

It's important to note that this does not make poetics a sort of complement or supplement to science. Poetics does not become the aesthetic overlay or communication device of science. As mentioned previously, it makes poetics epistemically different in its vision of the world. It also does not mean that poetics cannot be scientific nor science poetic. Science, and all of its "coherence and effecity" can be as poetic as "beauty". It is in the relationship between the Self and World, the process and intentions, the reader and the object, that produce the poetic state.

An illustrative example is that of the volcano. In science, volcanoes are phenomena studied via vulcanology. In geology, they are used as petrified archives of the world's procession through time; lithically ossified weather patterns, fertility readings, and climate snapshots. Each one remains an isolate, a vignette, a disjointed chapter. In poetics they are also ancestral and spiritual phenomena (of the communities surrounding them). They are threats to mortality and symbols of such threats. They are producers of fertility and symbols of such fertility. They are also the strange antinomy of this threat-fertility duality. They are revered as time recorders: stony layers of history. Here, in this last remark, we see the intersection of the poetic and scientific. Both are real and valuable, but only one preserves existential density and only one is able to hyper-specialise and materially explain.

Poetics then becomes a *mode of relation* and a *mode of attention*. It is how one chooses to look at and understand an object. But, also, how one positions oneself as an observer in the relation to the object of analysis. It re-encounters objects and reinscribes relations erased by neutrality.

2.2.4 Taḥqīq

> "Taḥqīq: in the technical language of scientists, it is ascertainment of an issue through a proof, just as scrutiny ("tadqīq") is the ascertainment of a proof through another proof [...] and among the recruiters of the Qur'ān, it is the accurate realisation of every single letter." (al-Tahānawī, 1996 as quoted in Pellò, 2023)

The concept of *taḥqīq* occupies a central position in Sufi epistemology, where it designates not the accumulation or transmission of knowledge, but its *realisation* and *verification* (Pellò, 2023). Commonly translated simply as "verification", taḥqīq marks a movement away from the inherited or imitative understanding (*"taqlīd"*) toward a *mode of knowing* grounded in direct engagement, perception, and phenomenological experience (al Obaidi, 2024, p. 117). Knowledge, in this framework, is not primarily representational. It is not something one possesses as an object. Rather, it is something one becomes *situated* through a sustained relation to what is being observed. Even from this brief introduction, we can already feel its coherence alongside the slowness of dwelling, for Bachelard, and the proactive maintenance of the poetic state, for Césaire.

A contemporary articulation of this convergence appears in Yusser al Obaidi's *Let Us Meet in the Margins [Barzakh]* (2024). Drawing on the concept of the barzakh, al Obaidi (2024) describes a liminal space that does not simply divide two terms, but simultaneously separates and connects them. The barzakh therefore permits a logic of neither/nor and both/and, in which difference is maintained without demanding resolution. In this sense, it resonates strongly with Césaire's poetic state; the poetic does not dissolve the antinomy between Self and World, but inhabits the charged relation between them (al Obaidi, 2024, pp. 117–121).

Al Obaidi (2024) extends this relational logic through taḥqīq, which she

frames as a process of embodying or realising meaning rather than merely representing it. Poetic knowledge therefore becomes less a question of identifying poetic properties than of cultivating a particular way of sensing and being in relation to the world. Her account provides a useful contemporary bridge between Césaire's poetics and the epistemology of taḥqīq developed below. Both refuse detached mastery in favour of situated, participatory forms of knowing in which the observer is implicated in what becomes knowable (al Obaidi, 2024).

Taḥqīq does not oppose rational inquiry or empirical attention. As Indian scholar Muhammed 'Alī al-Tahanāwī described in his 18th-century encyclopedia, *Kashshaf istilahat al-funun wa-l-'ulumn* (Glossary of Arts and Sciences Terms), seen in the opening quote, it is a scientific perspective that marries gnostic truth with deductive principle (al-Tahānawī, 1996 as cited in Pellò, 2023). It presupposed a form of disciplined observation. It resists epistemic neutrality.

> "This method, I argue, is at once poetic and philosophic, and is explicitly related in both directions, to a clear idea of 'imagining newness' [...], the peculiar conceptualising of a new '(poetic) style of knowing' in the later Mughal period." (Pellò, 2023)

To know through taḥqīq is to accept that the observer is always implicated in the act of knowing, and that truth is not exhausted by description or abstraction. Here, verification is not the confirmation of a proposition against an external standard, but the alignment of perception, understanding, and being (Pellò, 2023; al Obaidi, 2024, p. 117).

Pellò (2023) explores the poetic opus, *Ṭūr-i ma'rifat* ("The Sinai of Knowledge"), to explicate this concept. Ṭūr-i ma'rifat is composed of poetic descriptions of both natural and meteorological phenomena observed by Mīrzā 'Abd al-Qādir Bīdil. Bīdil opens the work invoking taḥqīq, announcing it as the methodological framework of the piece:

> āghāz-i bayān-i sayr-i bayrāt
>
> taḥqīq-i sawād-i qudratāyāt
>
> [This is] the beginning of the elucidation of Bairat's exploration,
>
> the taḥqīq of a land where the signs of potency (qudrat) appear

(al-Qādir Bīdil, 1962–1965 as cited in Pellò, 2023)

Pellò (2023) contextualises much of the language, explaining that the poem declares itself as a reflection 'realisation' of the landscapes witnessed through its *'sayr'*, meaning 'exploration' but also a word used for the *study* of texts (Pellò, 2023). In the second hemistich, the word *land* ("sawād") is also a *'scripture'*. This employs a common īhām double-entendre, which Pellò (2023) explains is further reinforced by *signs* ("āyāt", a reference to verses of the Qur'ān) (Pellò, 2023). The landscape becomes metaphorical

text to be *studied*, and from which the divine and embodied elucidations can be understood Taḥqīq becomes the method to *read* the unfolding experience of the natural world of God but, more importantly, to textualise that reading *truthfully*.

What we witness here is an exercise much like one conducted previously in reference to the volcano. However, with the discussion of the volcano, employed to elucidate the distinction between science and poetry according to Césaire, what we see here is a marriage of the two. For Bachelard and Césaire they remain disjointed. While they do well to explain their importance, authority, and form, they leave us with a separate practice. Taḥqīq marries the poetic and scientific, giving us a practice that can execute (in part) both, without compromise.

Truth, and knowledge in this instance, is often interchanged with the word *science* ("ma'rifat"). The natural landscape *is* knowledge. There is no understanding separate from experience. The poet-scientist's textualisation, taḥqīq, vocalises the mountain's *meaning* ("ma'nī") (Pellò, 2023).

In the work of Mīrzā 'Abd al-Qādir Bīdil, natural phenomena are not treated as inert objects to be classified, but as sites of unfolding meaning. Observation is inseparable from imagination, and imagination is not opposed to rigour. Nature is read as a text, but not one that passively yields meaning; its intelligibility emerges only through a disciplined, attentive *gaze* ("nigāh") that is willing to depart from inherited categories. For Bīdil, this encounter is first initiated by acute observation, which elucidates a departure from 'inherited knowledge' (Pellò, 2023) or that too often institutional knowledge imparted, or imprinted, hierarchically and linearly. We then see the importance of that aforementioned *gaze* ("nigāh"), as its initial direction becomes the locus of this poeto-scientific understanding.

This understanding of knowledge still finds a striking resonance with Aimé Césaire's account of the "poetic state". For Césaire, poetics does not resolve antinomies such as Self and World, subject and object, or interior and exterior. Instead, it inhabits these tensions and transforms them into lived intensity. New knowledges and meanings are discovered, facilitated, generated, through the marriage of maintenance of these antinomian equilibriums. The poetic state is not a synthesis, but a charged relation that must be actively sustained. Knowledge, in this context, is then not a neutral explanation but a situated encounter. Poetics becomes a mode of attention, a direction of gaze, through which the subject is re-established in relation to the world, rather than placed at an analytical distance from it.

Like Césaire's poetics, Taḥqīq shares this refusal of resolution. It does not seek to collapse difference into unity, nor contradiction into coherence. The distinction between knower and known is neither abolished nor stabilised; it is continuously negotiated through perception and engagement (Pellò, 2023; al Obaidi, 2024). Knowledge, here, is not static, but a process. It must be enacted, again, much like Césaire explains in his First Proposition (Césaire, 1990). Pellò (2023) points out that the "ordinary language, or, better, the reading of the world through the filter of acquired semiotic schemes" is actually a barrier to being able to understand it on this alternate

epistemological level (Pellò, 2023). This echoes Bachelard's discipline of unknowing or the capacity to forget knowing in order to transcend the limiting and restrictive categories we accumulate over time. "Taḥqīq is realised in the open "eye," internalised as the "mirror" of the heart" (Pellò, 2023). Both then become a call to engage the world intentionally, not through the one-dimensional encyclopedic perspective dictated by semiotic and institutional frameworks, but through *seeing*, or, perhaps more aptly, *reflecting*, within the dialectic matrix of ontology, gnoseology, and phenomenology; this is where we find poetics.

This framework offers a productive lens through which to engage computation poetically, without reducing code to its material inscription or treating it solely as linguistic artefact. Taḥqīq invites a different orientation. This is where metaphor becomes central, not as a descriptive overlay, but as a mode of epistemic relocation. Unlike the poetics described by Aristotle which foregrounds the ornamental, especially in its application to metaphor, we understand to engage code metaphorically, in this sense, is not to aestheticise it, but to shift how one attends to it. Metaphor allows computation to be encountered as process, rhythm, or transformation rather than as a static instruction. It frees us from the materiality of code and restores Césaire's concept of *mobilised richness*. It opens a space in which the observer can inhabit the antinomies of abstraction and execution without resolving them.

What is at stake in this poeto-scientific taḥqīq is not metaphor as ornament, but metaphor as epistemic operation. Metaphor functions here as a reorientation of attention, allowing phenomena to be encountered anew, outside of pre-codified interpretive schema. A poetics of computation grounded in taḥqīq does not oppose scientific rigour. Nor does it subordinate poetics to explanation. Instead, it articulates a mode of knowledge in which meaning is realised through situated engagement. Code becomes a site where poetic and scientific epistemologies intersect, not by function, but by shared commitment to realisation over representation. As with Césaire's poetic state, or with Bachelard's dwelling, this mode of knowing, of attention, must be actively sustained. It is not guaranteed by form alone, but emerges through attention, practice, and relation.

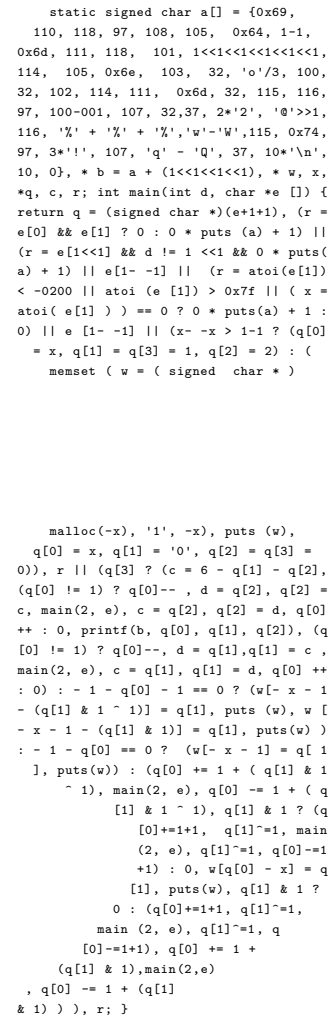
2.2.5 A Working Poetics

As we explore these definitions of poetics, we begin to understand the heterogeneity of it. The absence of any monolithic structure. Some participants may find poetics in the grace and elegance of an algorithmic solution. They may find beauty in a more formal poetics of efficiency and logic while, at the same time, rejecting the expressive aspect of poetics that links code to authorial subjectivity. This research does not value one form of poetic appreciation over the other, instead it looks at the valuation itself as valuable. This research seeks to map these varied and diverse poetic landscapes and the plurality of poetic ways of knowing that may coexist or conflict during an engagement with code text.

What emerges from these converging traditions is a conception of poetics

not as genre, ornament, or aesthetic classification, but as a disciplined mode of attention (al Obaidi, 2024, pp. 115–120). Poetics is neither reducible to metaphor nor confined to (literary) artefact, but explored through them and springs from them, when combined with the mobilised richness of human experience, the phenomenological depth of being, and the sustained engagement of attention; we begin to see that "there are poetic ways of seeing which transcend the function of our eyes. [...] It is a liberation of the senses." (Gowen and al Obaidi, 2024, 24:20) It is a relational operation. Meaning is not decoded or stabilised, but realised through inhabitation. In this sense, poetics is epistemically distinct from scientific explanation, not because it rejects rigour (indeed our poetic interpretations *should* be rigorous), but because it refuses *reduction*. Neither of these epistemologies are superior and therefore neither are worthy of any ontological priority. Instead, the reader should approach the object open-minded with a thoughtful reflection on positionality and an acute awareness of goals and intention. Poetics does not isolate the object from observer, but implicates the observer within the act of knowing. Poetic knowledge emerges where antinomies are held rather than resolved, where attention becomes participatory rather than extractive.

When we examine, for example, the code text *"Puzzles"*, by Jay Berkenbilt (1993), we are presented with code in an unusual form; in the shape of a semi-colon:



```
    static signed char a[] = {0x69,
  110, 118, 97, 108, 105,  0x64, 1-1,
0x6d, 111, 118,  101, 1<<1<<1<<1<<1<<1,
114,  105, 0x6e,  103,  32, 'o'/3, 100,
32, 102, 114, 111,  0x6d, 32, 115, 116,
97, 100-001, 107, 32,37, 2*'2', '@'>>1,
116, '%' + '%' + '%','w'-'W',115, 0x74,
97, 3*'!', 107, 'q' - 'Q', 37, 10*'\n',
10, 0}, * b = a + (1<<1<<1<<1), * w, x,
*q, c, r; int main(int d, char *e []) {
return q = (signed char *)(e+1+1), (r =
e[0] && e[1] ? 0 : 0 * puts (a) + 1) ||
(r = e[1<<1] && d != 1 <<1 && 0 * puts(
a) + 1) || e[1- -1] ||  (r = atoi(e[1])
< -0200 || atoi (e [1]) > 0x7f || ( x =
atoi( e[1] ) ) == 0 ? 0 * puts(a) + 1 :
0) || e [1- -1] || (x- -x > 1-1 ? (q[0]
  = x, q[1] = q[3] = 1, q[2] = 2) : (
    memset ( w = ( signed  char * )
```



```
    malloc(-x), '1', -x), puts (w),
  q[0] = x, q[1] = '0', q[2] = q[3] =
0)), r || (q[3] ? (c = 6 - q[1] - q[2],
(q[0] != 1) ? q[0]-- , d = q[2], q[2] =
c, main(2, e), c = q[2], q[2] = d, q[0]
++ : 0, printf(b, q[0], q[1], q[2]), (q
[0] != 1) ? q[0]--, d = q[1],q[1] = c ,
main(2, e), c = q[1], q[1] = d, q[0] ++
: 0) : - 1 - q[0] - 1 == 0 ? (w[- x - 1
- (q[1] & 1 ^ 1)] = q[1], puts (w), w [
- x - 1 - (q[1] & 1)] = q[1], puts(w) )
: - 1 - q[0] == 0 ?  (w[- x - 1] = q[ 1
  ], puts(w)) : (q[0] += 1 + ( q[1] & 1
      ^ 1), main(2, e), q[0] -= 1 + ( q
            [1] & 1 ^ 1), q[1] & 1 ? (q
               [0]+=1+1,  q[1]^=1, main
               (2, e), q[1]^=1, q[0]-=1
               +1) : 0, w[q[0] - x] = q
             [1], puts(w), q[1] & 1 ?
          0 : (q[0]+=1+1, q[1]^=1,
        main (2, e), q[1]^=1, q
      [0]-=1+1), q[0] += 1 +
    (q[1] & 1),main(2,e)
 , q[0] -= 1 + (q[1]
& 1) ) ), r; }
```

This imagery is immediately rich. From a technical perspective, we recognise the semi-colon in C as the symbol to indicate the termination of a statement. However, in literature, the semi-colon is not a strict termination but a point were termination *could have* happened and the author decided to continue with a distinct yet related statement (You may find many examples of them throughout this paper; we are quite fond of them). In code, the succeeding statements need not be related (though they tend to be due to the natural way an author produces a linear procession of thoughts; proximity often implies some relation, but it does not necessitate it). So, then, what can we make of the semi-colon here? Is there space for this literary understanding of the semi-colon? Is there a relationship between what the semi-colon means in code and in literature?

Berkenbilt (1993) explains in the description of this IOCCC[12] entry, that the imagery was employed as means to explore the question *"what is a line of code?"* (Berkenbilt, 1993) He explains that simply counting semi-colons will leave the investigator to an impasse between counting lines without definitive terminators and logically connected lines which *do* contain them. How many semi-colons are in this code snippet? Two? Three? How many lines of code are there then? Does the semi-colon here operate, in fact,

[12]International Obfuscated C Code Contest

more in line with the literary writer's, where they are ambiguous in their declaration of termination and instead are a bounding and logical parcel, a delineation of a larger statement, or idea, that pre-exists the semi-colon and persists after it? The code becomes the semi-colon; but to what statement? What is the narrative or logical line being punctuated? Is it Berkenbilt's perspective on code itself and this small moment acts as an end-statement? No, it acts as the *potential* to end being ignored. The idea persists beyond the semi-colon, the literary semi-colon, where there is in fact an unspoken connecting concept we must wait to read to finish the thought.

Under this working definition of poetics, the kind of rumination above is given life in a way the teleological engagement would not sustain. Code need not be declared a poem in order to be encountered poetically. The obfuscated C code entry, with its explicit aesthetic self-consciousness and deliberate opacity, already invites such an engagement, foregrounding form, and authorial gesture.

Code Europa's *"Website Evidence Collector"* (see 4.2), by contrast, appears primarily instrumental. It classifies hosts, distinguishes first and third parties, aggregates and orders digital traces. Yet, precisely in these acts of classification, ordering, and naming, relational structures are produced that say something about the world in which this automaton exists, and also about the world the programmatic architects live in.

In the constructor of the Inspector class, we are presented with a quite traditional procession of the class' instance variables:

```
class Inspector {
  constructor(
    private collectorOutput: CollectorOutput,
    private firstPartyChecker: FirstPartyChecker,
    private cookieRecorder: CookieRecorder,
    private beaconRecorder: BeaconRecorder,
    private hostRecorder: HostRecorder,
    private webSocketRecorder: WebSocketRecorder,
  ) {}
```

Traditionally, this is unworthy of much attention beyond teleological questions: are these constructed correctly? What do they contain? What will they do? But, what if we reflect on the very nature of enumeration and lists conceptually?

Paolo Zellini (2020) writes in his book, "Mathematics of the Gods and the Algorithms of Men: A Cultural History":

> "The process of enumeration employed since the most ancient times often had the purpose of making things [...] real by means of demonstration of divine gesture.
>
> Enumerations, census, lists and catalogues recur frequently Homer and Hesiod, Aeschylus and Herodotus, as well as the Old Testament. Enumeration was the prerogative of lógos." (Zellini, 2020, p. 18)

What, then, is made real by listing these? A reality is defined and demarcated. An authority is then produced; a hierarchy of world-definer and world-inhabitant. These variables become pillars of attention. A universe defined by collecting, checking, and recording. Nothing should be produced, only gathered and investigated. This sense of cosmic order is further reinforced by the rhythm with which the names can be read:

```
class Inspector {
  constructor(
    private collectorOutput: ...
```

The deep, long assonant *'o'*s and the rhyming *'ector'* sounds act almost like a drum, heralding the declaration of something important. Like a royal decree or a command to attention: "The laws of the land are about to be announced." The siren sound of the world you are about to enter.

Under poetic attention, these structures may be inhabited rather than merely executed. In both cases, poetics does not reside intrinsically within the text, but emerges through the mode of attention brought to it.

This working poetics, therefore, establishes the conceptual ground for engaging code not solely as an instruction or optimisation problem, but as a site of relational realisation.

2.3 Close Reading

Having established poetics as a mode of attention grounded in sustained, situated, and non-reductive engagement, the methodological question becomes how such a relation might be practically facilitated. We therefore turn to close reading, not simply as a literary technique, but as a practice whose emphasis on slowness, attentiveness, rereading, and interpretive plurality offers a means of operationalising the poetic framework developed above. Collective close reading provides the bridge between the epistemological commitments of this section and the empirical investigation that follows.

Close reading grew in popularity in scholarly literary analysis in the mid 20th century (Hinchman and Moore, 2013). Popular texts that developed the practice include "How to Read a Book" (Adler, 1940) and "How to Does a Poem Mean" (Ciardi and Williams, 1975). However, today, the title "close reading" has come to encompass a number of different literary practices aligning on their devotion to the interpretation of (usually textual) artefacts (Rabinowitz, 1992). Rabinowitz (1992) describes close reading's position in the school of cultural and literary analysis as the 'nearest thing we have to a shared principal' and quotes Jonathon Culler in describing it as 'the requirement' for New Criticism (Rabinowitz, 1992).[13] In general, close reading describes the process of engaging a text with a level of sustained, slow attention, with the goal of near-exhaustive interpretation on a textually microscopic scale; i.e. each sentence or even word.

[13]Rabinowitz was critical of this status, with the central thesis of this article being a call for other methodologies.

In “A New Handbook for Literary Terms”, Mikics (2007) describes close reading as “investigat[ing] the specific strength of literary work in as many details as possible. It also means understanding how a text works, how it creates its effects on the most minute level” (Mikics, 2007, p. 61). For the creators of close reading, the New Critics, this method was a way to teach students how to first unread during initial exposure to a text, and, only then, continuing on to reread, now with a deeper understanding that was originally obfuscated by their own prejudices and ignorances (Bialostosky, 2006). Close reading is “productive attentiveness” (Bialostosky, 2006).

What we already begin to see is that close reading emphasises aspects that are central to the working definition of poetics developed for this investigation. The acts of slowing down and examination of the minutia mirror the dwelling and over-analysis of Bachelard. The foregrounding of attentiveness is aligning with the significance of gaze and mode of attention we see in Taḥqīq and Césaire. Bialostosky’s (2006) reference to the act of unreading and rereading is particularly noteworthy to us in context of this investigation which demands the deconstruction of pre-existing understandings of how to engage with code text in order to make way for the reconstruction of a perspective that accommodates traditional teleological understandings, alternative non-teleological understandings, and their intersection and interactions. Close reading, no doubt in being a tool developed for the analysis of poetry, is perfectly aligned with the conceptual framework in which we are working.

The authorial New Critics posited close reading as a ‘rigorous objective method for extracting the correct meaning of a text’ through the meticulous analysis of patterns, language, and metaphor/symbolism (Hinchman and Moore, 2013). However, subsequent literary theorists have adapted close reading to a method that accommodates plurality in sense making of texts (Hinchman and Moore, 2013). Reader-Response theorists encourage a more intersectional and dialogic employment of close reading that would acknowledge the role of the reader’s personal background or experience in the development of meaning or the contextual interpretive community in which the close reading is being employed (cautionary close reading). We can see here parallels to Roland Barthes (1967) wherein authority is redirected toward the reader in the production of meaning rather than the reader as an investigator for meaning produced by the author. You also have critical theorists highlighting the positionality of a reader in relation to text as a complex dynamic which is influenced by race, sex, gender, class, and other social markers (cautionary close reading), and deconstructionists, who follow from Derrida (1997), who consider words to be absolved of fixed meaning, allowing for the uncovering of a text’s multiple, often radically contradictory, meanings (Derrida, 1997).

To provide a guide to how we have come to understand close reading practically and implementably, in the context of this research, we outline key aspects of close reading, gathered by reading a number of texts on how to conduct close reading (Fisher and Frey, 2012; Frey and Fisher, 2013; Hinchman and Moore, 2013; Kain, 1998; Payumi and Hartati, 2018):

1. Annotate: Read with a pen in your hand at all times. Underline, circle, draw arrows, connect things, make notes, interact with the text by hand. This is making you an active reader rather than a passive observer. It forces you to apply gaze, direct attention, and be intentional with engagement.

2. Short passages: Close reading is not about getting through as much as possible but understanding as much as possible. Like Bachelard, we slow down and dwell within the words. We read line-by-line, pausing and reflecting, annotating and considering. Sometimes we may read more and sometimes even less; but pacing should be slow.

3. Reread: Rereading is also a means to affix gaze and dwell.

4. Look for patterns: Repetition, contradiction, similarities across the text, these all become important indicators of meaning that should be identified and investigated.

5. Ask why and how: Every pattern located should be questioned, particularly with questions of how this pattern is created and enforced and why this pattern is appearing at all.

6. Consider the author: Following from this last point, here we consider the author. What are their intentions with this? Why would they use this particular form? From where are they writing this?

While often framed as an individual interpretive act, a substantial body of research emphasizes that close reading is equally effective as a *social* practice, in which understanding emerges as a dialogue and through collective sense-making. Pedagogical research and reflections on close reading highlights the importance of rereading, annotation, and text-dependent questioning (Bruder, 2020; Fisher and Frey, 2012; Payumi and Hartati, 2018), but also stresses that discussion among readers plays a central role in how meaning is constructed and refined (Fisher and Frey, 2012; Izzo, 2013).

Close reading's process of going line-by-line, pausing frequently to reflect and discuss, and emphasising zooming into terms and phrases resists and facilitates the non-normative engagement of code we hope will lead to a departure from teleological, instrumental focus. Making them collective creates space for intersubjective meaning generation. Ideas, interpretations, and opinions circulate and interact giving space for us to analyse these interactions and draw conclusions from them. The collective close readings becomes a centre for social politics which is important in seeing how the backgrounds of participants (technical vs. non-technical, for example) influence the ideas accepted and challenged and how these prejudices or patterns change over sessions.

Collective close readings then become a logical choice for this poetic investigation seeing as they are used currently in literary and cultural analysis to examine poetic dimensions of text, but the collective aspect means they present an opportunity to study intra-group dynamics such as hierarchy, thematic development between interlocutors, and the interactions between

opposing ideas. They create the precise conditions for the emergence of the dimensions of poetic understanding described above in a manner that makes them measurable. This allows us to more accurately observe whether epistemic shifts have transpired.

It is worth mentioning that collective close readings may be compared to an existing practice in computer engineering: collective code reviews. An important differentiator is that, unlike code reviews, collective close readings do not aim at optimisation, correction, or teleological understanding of the code. They don't seek to adapt or alter the code in any way. Ambiguity is generally not tolerated in the code review, but rather seen as something to exterminate, whereas in collective close readings it is expected. In the close reading speculation is invited and also sustained, in code reviews it may be offered but must then be investigated and either falsified or proven. Disagreement is treated as analytically productive rather than something to be resolved in a collective close reading. The goal is not to stabilise meaning, but expand the interpretive space surrounding the code. Code reviews are normative and convergent, close readings are exploratory and divergent.

3 Research Question

Following from this understanding of what is poetic, it leads us to an important realisation about code texts as literary objects. We are not claiming that code *can* be poetic, from our previous definition this claim becomes trivial. We claim that different textual forms afford different poetic dimensions and that the textual form of code is currently under-investigated and therefore under-theorised. The register or framing within which this research is positioned is as an epistemological and methodological investigation; we are exploring what can be known about code and how alternative knowledges can emerge from alternative engagements. Our contribution is not "code has poetic potential" but rather, "what kinds of poetic dimensions does code make available when we approach them through a poetic lens and how do readers come to discover, share, and articulate them?" We seek to explore under what conditions can poetic dimensions of code become perceptible, articulable, and discussable. Furthermore, we hope that by expanding the landscape of ways to engage with code into the realm of the poetic will influence the way code could be produced, an epistemic shift in relationality that is bilateral.

To remain philosophically consistent with the nature of our own definition of poetics, given that poetics as a mode of attention or an interobjective and intersubjective relationship, it must be acknowledged that we cannot make generalisably descriptive statements about some *formal description* of the poetics of code texts. Any poetic characteristics of the code uncovered in this research are still valuable objects for analysis, but only insofar as they reveal the manner in which the text was engaged with by the readers. Explicit poetic techniques and dimensions will be a means to understand how the participants related to the code in this new way and indicators of whether the alternative engagement was successful in elucidating an alter-

native epistemology of code.

This means the focus of the study is *how* the participants engaged with code and whether the method employed produces a desired epistemic shift in perspective. The primary object of analysis becomes the reader-code relation under a poetic frame and the effectiveness of the method employed to generate this frame. Poetic features are empirical traces of that relation, not universal claims. Other empirical traces include (but are not limited to) the language used by participants, inter-group dynamics, the kinds of knowledge extraction strategies employed by participants, and any shifts over time in these.

With these aspects of what we are looking for, we have arrived at the research question:

> *What is the effectiveness of collective close reading as a method for engaging with code texts in producing an epistemic shift from teleological interpretation to poetic modes of understanding?*

A poetic mode of understanding in this context, following from our definition derived from our previous analysis of Aristotle, Gaston Bachelard, Aimé Césaire, Muhammed ʻAlī al-Tahanāwī, and Mīrzā ʻAbd al-Qādir Bīdil would be a suspension of teleology, an increase in tolerance of ambiguity and mutually coexisting interpretations, the acceptance and legitimisation of speculation, and a re-structuring in the hierarchy of expertise. Furthermore, "effectiveness" in this context does not refer to measurable improvement against a control condition. Rather, it refers to whether collective close reading produces identifiable qualitative shifts in discourse, interpretive authority, and epistemic framing. Crucially, this effectiveness is not contingent on a participant's explicit acknowledgement of the experience as 'poetic'. While they may reject the label, or use other terms to describe their engagement, the research will analyse their behaviour, attentiveness, and language during discussions to diagnose a poetic mode of understanding. While an explicit awareness or self-diagnosis would be ideal, other indicators such as tolerance for ambiguity, speculative inquiry, and a shifting away from instrumental logic may manifest, displaying the epistemic expansion alluded to in the research question. Our role is to observe these behaviours as indicators of a poetic mode of understanding, even in the absence of the participant self-identification with the term.

Furthermore, there is also an important sub-question:

> *What are the poetic features commonly identified and discussed when poetically engaging with code text?*

While we are not searching for an exhaustive nor generalisable list of poetic features to describe codes poetic dimensions, consistent precipitation of common features does indicate congruity in the poetic relations between readers and code text. Understanding these dimensions, features, and congruities is important for understanding the ways in which readers may develop this poetic relationship with code. It may also serve to inspire future

methodologies for poetic software interpretation or for understanding technical qualities of code, for example in how we see Exman and Plebe (2016) entertain the idea of poetic metaphor and metonym to derive a proposal for a novel software structure (Exman and Plebe, 2016). An aggregation of these features may illustrate new directions for further investigation in how to develop methodologies for poetic engagement of code texts (as well as other technical artefacts) and teach us something novel, unexpected, and valuable about the nature of code, as a textual material, itself. These features are not exclusively isolated to what we understand as 'traditional' poetic features either; things such as rhyme, metre, or metaphor. Participants may find poetic value in the technical features themselves. Understanding this nuance is done through analysis of the language they use to discuss such features. In the same way the literary analysts may dedicate great spans attempting to understand the *poetic reason* behind a creative liberty taken in a poem, a decision for one poetic device over another, it is feasible that participants may employ a similar process to understanding the *poetic reason* for the selection of one algorithmic strategy over another, or one data type or another; the technical features may *become* poetic.

Poetic interpretations are not rigidly concerned with features like optimisation, validation, and efficiency. That is not to say these features in the code cannot be poetic. Analysing when and where these coding metrics are employed or sacrificed can be rich and valuable for poetic interpretation, but unlike dominant ontological analysis of code production these features are not considered the *end goal* from a poetic perspective. Poetic engagements tend to produce multiple interpretations, which may even potentially be contradictory to one another, however they can co-exist mutually as equally legitimate understandings. This openness to a multitude of interpretations of text is something not tolerated in a traditional method of understanding code which seeks to converge on a singular definition of a code's purpose or meaning. This feature would manifest as a tolerance for divergence in participant interpretations rather than a consistent return to attempts to converge. Speculation on features of a piece of code (such as the author's intentions and perspectives or metaphorical subtexts) is something that differentiates the current domains of poetics and (computer) science. Within the former, this practice is seen as central, in the latter there is little, if any, tolerance for it at all. Speculation being explored deeper, being sustained within discussions without collapsing to ground-truth seeking, is a sign of the epistemic shift from one domain to the other. Finally, in a typical iteration of code analysis between technically capable individuals and non-technical individuals, one would expect a level of analytical authority to implicitly and organically emerge on behalf of the technical persons, considering their more reliable ability to understand the code in a traditional sense (explain its function, point out errors, ideate improvements, and so on). However, in a setting that tolerates poetic dimensions, then technical know-how is only one strategy for engaging with code text and there are now others that can be fulfilled by people who do not have technical expertise, and they may be able to extract latent poetic meaning even better than a technical expert,

and, of course, there are individuals who can, either alternate between or occupy simultaneously, both perspectives.

Within this framework we are not attempting to add a layer of poetry onto code, but instead we seek to witness an actual evolution in the way the text of the code itself is thought of and interacted with. We are not attempting to *convince* a group of participants that we can trivially compare poems and code, but instead we are attempting to create an environment where what the code represents and how it can be understood begins to expand. This is because what we are seeking is a latent poetic dimension *inside* the code texts rather than the ability for a group of readers to arbitrarily apply poetics to the code which is a trivial and uninteresting statement. Our ambitions are not aesthetic, in the sense that we would want to find rhyming features or some other superficial dimension of an aesthetics of poetry (albeit these may reveal very interesting things about the code), instead, the goals are epistemic. To destabilise dominant ontologies of code as purely deterministic, technical, and objective artefact.

The posited theoretical framework does not make any claims about *which* poetic features or dimensions will appear, but instead the conditional claim, "if code is linguistic, poetics is a universally applicable mode of attention, and linguistic practices are rich in poetic attention, then a poetic dimension of code is not categorically excluded." Here, we establish the plausibility of the poetic engagement of code and as such failure of the study itself does not falsify the hypothesis' plausibility, but instead raise questions about the condition under which such engagement can be facilitated. As such, the research is also deeply methodological. We are investigating one way in which the emergent capacity of code could be facilitated, namely *collective close readings*. The goal of this research is indeed to search for this epistemic shift in code relation, but the manner in which it is being produced is specific to the method, making the collective close reading a locus of attention for variables of success and failure. Failure to discover epistemic shifts should inspire interrogation primarily into the design of the collective close reading or the choice of this methodology itself, not to immediately invalidate the theoretical motivation. Success to discover the shift also means we have discovered one suitable methodology for making available this alternative dimension. There may indeed be others that are more effective or suitable for specific sub-domains of poetic analysis.

Pedagogically and politically, this research broadens the field of who can meaningfully engage code. By challenging the assumption that understanding code is contingent upon technical mastery (and technical mastery alone), we de-monopolise and destabilise hegemonic structures which dictate who can produce code, what code is valuable, and, perhaps most importantly, what makes code valuable. By emphasising interpretive, speculative, and dialogic modes of engagement, collective close reading repositions code as a text that can be approached critically by participants with diverse backgrounds. This redistribution of interpretive authority disrupts techno-objectivist narratives that frame code as neutral, opaque, or accessible only to experts, and instead positions itself as a situated, authored, and

value-laden artefact. In doing so, this work opens space for interdisciplinary pedagogy and critical literacy that engage code not only as a functional tool, but as a cultural and expressive medium.

4 Methodology

This section outlines the practical implementation of the research design. We begin by introducing the collective close reading sessions, followed by a justification for the selection of specific code texts. We explain why these texts were chosen not for an a priori 'poetic-ness' but for their interpretive affordances. Finally, we detail the data collection process, including the participant selection and the timing of the exit interviews.

4.1 Structure

Two experiment trials (henceforth referred to as Trial 1 and Trial 2) of three meeting sessions each were conducted. Each of the three sessions were approximately two hours in duration. From our close reading research, longer sessions are preferred to facilitate re-reading and slow reading. However, one long six hour session runs the risk of fatiguing participants. Yet, at the same time, we wanted single session durations long enough to sustain engaging conversations. From experience conducting and participating in close readings in past, also taking inspiration from common lecture durations, we concluded two hours is a reasonable length for a session. Three sessions were chosen because this seemed a reasonable demand of commitment from participants while providing enough sessions to extract valuable content for analysis.

Each session of a respective trial was populated by the same participants, absences were tracked and were considered and discussed during analysis. The groups were composed of a mix of 2-3 technical and non-technical persons each and one facilitator (a member from the research team acted as the facilitator). The technical consideration allows us to examine the intergroup politics and investigate the initial distribution and evolution interpretive authority. The relationship between technical and non-technical participants and those same participants and the texts is an important aspect to seeing the relationship between epistemologies of computer science and poetics. A group population size of 4-5 was chosen because of the presence of only a single facilitator per trial group. From experience, with one facilitator, more than 5 participants could become difficult to manage. Such a small discussion group size also creates a more intimate environment where each individual has more of an opportunity to speak. However less than three rules out any ability to study intra-group dynamics.

Each trial ran in parallel, each with different people but the same texts. Two pieces of code were analysed over the course of the three sessions (the content of the code is discussed in the succeeding section, 4.2). Consistency of texts allows to control better for social aspects; we can be more certain that arising disagreements, conflicts, congruities, and other patterns

or anomalies arise from the group or epistemological frictions and dynamics rather than text content.

Both trials were conducted with groups composed of university students. This specific demographic was chosen simply due to accessibility. There was an attempt to conduct a third trial composed entirely of professors and senior researchers, however conflicting schedules proved too incompatible to organise such long sessions within an acceptable time frame. We decided not to mix student and researcher groups in order to control for the influence of external hierarchy. This preservation of horizontal hierarchy means the only authoritative differences were with disciplinary experience rather than institutional culture.

The facilitator's role is to give context to the code, explain format and structural questions about close reading as a practice, help facilitate or stimulate discussion, ask probing questions, and maintain a constructive, healthy group dynamic. The facilitator's role does *not* include guiding the conversation toward any particular direction or offering solutions or resolutions to conflicts over interpretations. The facilitator can indeed offer their own opinion on their interpretations of particular excerpts, however they should be offered as a means to inspire discussion flow and development, not as a means to dictate group sentiments. The facilitator can intervene in conflicts within the discussion that seem to be persisting, but that interference should not be alignment with one position over the another. Instead it should, as before, be to keep the conversation progressing constructively rather than becoming stagnant. The facilitator is also responsible for familiarising the non-initiated with close reading as a practice.

The facilitator should not, however, be understood as a neutral observer external to the interpretive environment. The facilitator was also involved in the theoretical development of the research, the design of the close-reading protocol, the selection of texts, and the subsequent analysis of the sessions. Moreover, the semi-structured character of the sessions required active intervention through probing questions, requests for elaboration, technical clarification, and occasional attempts to revive or extend discussion. The interpretive field observed in the sessions was therefore partly scaffolded by the research design itself.

This limits the extent to which individual participant remarks can be treated as spontaneous evidence of an independently occurring epistemic shift. In analysis, greater evidential weight is therefore given to interpretations initiated by participants without direct prompting, to interpretive behaviours that recur across participants or sessions, to changes in behaviour over time, and to moments in which participants extend, resist, or transform the framing offered by the facilitator. Facilitator-elicited responses remain relevant, but are interpreted primarily as evidence of what the method made possible rather than as evidence that the interpretive orientation arose independently of the intervention.

Session 1 of each trial included a brief introduction to the experiment. Participants already received an explanatory document informing them of the structure and framing of the experiment so this content was already

somewhat familiar. This was so there was another opportunity to answer logistical, organisational, and structural questions about the experiment before beginning. This session also had a short introduction to close reading as a practice. This meant the inclusion of a short exercise where close reading was done on a more traditional text for 10-15 minutes before moving onto the first code text[14]. This first session only introduced and close read part of the first code text.

Session 2 began by continuing with code text 1 and discussion of which was finalised and concluded with within this same session. Afterwards, code text 2 was introduced and close reading of this text began. This split across the sessions is partly due to having to introduce close reading during the first sessions and having to take time to introduce and open each session. It also allows time to think about the groups' discussion of the text outside the context of the research and gives participants an opportunity to follow up on thoughts in the next session. Each session was semi-structured, so if participants felt very compelled to move on or end early, that was accommodated.

For both texts, a very general explanation of what the code does was provided; not specifics about each code block and method, but the general, high-level task the code is designed to complete. However, if participants asked, explanation of a particular block, method, or line was provided. This was so that context as to what the code does was provided in order to better apply close reading (context was one of the steps we previously discussed in our description of close reading, see 2.3), that was also why questions were answered. But a deep analysis was avoided so as not to set any tone of emphasis on technical purpose and execution.

The final session for each trial, session 3, continued with code text 2, concluding the discussion within that session. After finishing close reading both texts, there was an opportunity for group reflection on their sentiments toward the practice, insights into its effectiveness, reflection on any changes regarding their perspective on code (or lack thereof), and general discussion of how the trial went.

The trials were completed once all three sessions were concluded and each participant had taken part in an exit interview. These interviews were conducted privately in a one-on-one environment. They were also semi-structured interviews with some prepared questions, however the participant guided the discussion with their individual reflections. The reason for the private exit interview was to give the participants a more open and comfortable space to offer opinions they may have been uncomfortable sharing in the group environment. The goal was to discover their personal experience with the form of the trial and sessions, the content discussed, but also the social experience of applying the close reading collectively. To ensure the experience remained fresh in the participants' minds, these interviews were conducted within one week of the final session.

[14]The piece used was the poem *Silk Flowers* by Che Qianzi, translated by Wang Ping and Lewis Warsh, found in the anthology *New American Writing* issue 22 (Hoover and Chernoff, 2004).

4.2 Content

The code chosen for analysis was identified on the basis of its affordances for collective close reading, we did not choose these texts because we initially perceived them as 'poetic'; rather, we chose them because they appeared to possess multiple dimensions that promised to be fruitful for sustained interpretation; dimensions such as structural, lexical, or political depth. The poetic quality was thus treated not as a pre-existing label but as an intrinsic *potential* for meaning-making that the method aimed to activate.

Two contrasting cases of code texts were chosen to investigate poetic interpretation under different conditions. Both can be found in respective appendices.

> 'Adder', Heather Downs and Selene Makarios, retrieved from the Obfuscated C Code repository (January 27th, 2026). [15]

> 'Website Evidence Collector', Robert Riemann, Company BitnessWise, Roland Schilling, Company Avast, Per Ploug (Zalando), retrieved from Code Europa repository (January 2nd, 2026). [16]

Each participant was presented with an individual copy of the code, asked to read through it independently, and then engage in collective analysis. As per our close reading research, (see 2.3), they were encouraged to mark, make notes, highlight, and annotate as they saw fit on their individual copy, though a collective version was made visible to also track group discussion. Before beginning in analysis, as previously mentioned, a brief overview was provided as to what the code does, but on an abstract level. This contextualised the code but allowed space for interpretive autonomy of technical specificities, as mentioned in the previous section.

The code texts selected for this study serve as the object of analysis during the collective close reading sessions. The texts were never presented as having been selected for their being exemplary or representative capacity of software codebases in general. Instead, they were introduced as *text*, whose structure, language, and context afford sustained interpretive engagement. The selection process was guided by a methodological consideration rather than claims of aesthetic or technical significance.

When searching for code, there were a number of things to consider. Some of the stand out criteria that made these examples attractive candidates were their public accessibility and inspectable source codes. This ensured no intellectual property infringement or transparency issues with later discussion. Both pieces of code have well-defined contextual backgrounds and situated purposes. Rather than being a single-author project for an unnamed assignment found in the recesses of GitHub, the projects have well described background and motivation; one being a government

[15] https://www.cise.ufl.edu/~manuel/obfuscate/heathbar

[16] https://code.europa.eu/EDPS/website-evidence-collector/-/tree/main?ref_type=heads

reporting tool, the other being a competitive, aesthetic exploration. This offered good grounding and points of departure for interpretation. It also opened up discussion for positionality of the code and situatedness, leaving room for discussion on the institutional context and authorial role. The code has sufficient content for analysis while still being segmentable for passage reading, however, each in an individual manner. The first can be divided on the basis of form, rather than structure, and the second vice versa.

Adder is made of only one heavily obfuscated block. The code opens with a codex, translating segmented code lines into permutations of the word *'main'*, so the code is, in theory, understandable through re-substitution. However there are many different dimensions worthy of discussion, choice of problem, choice obfuscation method, choice of visual representation, the explicit comments of from the author, among others.

Website Evidence Collector is more traditionally divisible, being composed of multiple classes and blocks. We looked only at the *Inspector* class, which itself can be segmented into more atomic structures like methods or even individual expressions or statements. Finally, there was a consideration to the broader social and political concerns surrounding code.

In the case of *Adder* the foregrounding questions are epistemological and disciplinary. Using code as an meta-reflection through aesthetic exploration challenges much of the hegemonic expectations of how to ask questions about the nature of code, making the repository quite politically engaged. *Website Evidence Collector* is political in another sense, perhaps more directly. It is a coding tool used by the EU to examine web traffic making it directly implicated in a number of political concerns from surveillance and privacy to autonomy and security.

Furthermore, we believed presenting *Adder* would help to quickly illustrate that code can indeed be an expressive medium. It quickly deconstructs the barrier to entry for non-technical persons and opens the door to technical persons for engaging with technical material interpretively. Presenting the *Adder* text first also eases the participants into the process of transitioning from the introduction exercise of close reading traditional literary media to close reading highly technical non-traditional media (code). Both have laden political and epistemological dimensions that allow for divergence and breadth of interpretive potential.

The reason for choosing two contrasting texts is, as mentioned, to investigate the poetic potential under different conditions; conditions where explicit aesthetic and expressive considerations are employed, and those where they are perhaps not considered, and instead, are omitted in favour of rigid teleological ends. This helps to isolate collective close reading as the source of the poetic extraction during the sessions. In the case of both sessions producing valuable epistemic shifts and poetic feature descriptions, collective close reading would appear as an effective candidate at facilitating such poetic richness. Collective close reading demonstrates its ability to amplify pre-existing latent expressive potential and generate inter-subjective and inter-objective poetic relationality.

4.3 Analysis

Participants read code line by line, aloud, at a deliberately slow pace. Frequent pauses were taken to speculate on meaning beyond functionality, question features of the code such as naming, structure, ordering, or omission, and reflect on what assumptions the code makes about the world, users, or problems. Pauses occurred either when participants felt compelled to stop and ask a question or make a point, or after a few lines of code had been read that contained enough conceptual material for analysis. They were asked to treat each authorial decision as interpretable, even if that interpretation could later be questioned or disrupted. They were also encouraged to share personal associations, metaphors, or imaginative responses elicited by the code that encourage divergent interpretability. This was facilitated by each participant having a textually printed version of the code in front of them where they may underline, annotate, highlight, connect, and demarcate as much as they like to explore these interpretations. For close reading the goal is not to read a lot, but to read with great intention. Re-reading the same excerpt can be much more fruitful than getting through the whole text, especially for our purposes of studying dynamics rather than content.

During pauses the group collectively engaged with each other's shared ideas also. They were encouraged to probe one another's interpretations and suggestions, pushing back and critically challenging ideas, comparing and contrasting ideas. Ideally, when pushing back, they were not expected to do so to seek resolution but to either understand deeper or explore a theme more broadly. When comparing and contrasting ideas, we considered whether this was to align and settle on a singular interpretation or whether multiple, even contradictory readings were allowed to coexist.

The data collected for analysis consisted of sound recordings of the sessions. With this we analysed everything said during the sessions as well as the pauses and silences. The facilitator also took field notes throughout the sessions, recording observations on social dynamics, patterns in behaviour, and body language. The field notes also referenced who was annotating their individual sheets and how much, since we did not collect their personal copies of the texts. This was to give the participants a space to make comments and explore ideas without having to share out loud, encouraging engagement without fear of failure or embarrassment. A similar protocol was followed for the exit interviews wherein they had their audio recorded and hand-written facilitator notes were taken.

We distilled four distinct indicators to use as measures toward the efficacy of collective close readings at producing epistemic shifts from teleological to poetic modes of understanding. From the audio recordings and field notes collected from both the trials and exit interviews we looked for the following indicators:

1. Reconfigurations of how code is understood.
2. Expansion of interpretive authority.
3. Sustained speculative, non-teleological discourse.

4. Consistent emergence of common poetic features

The first refers to participants (especially with coding backgrounds) displaying movement from instrumental to expressive engagement with the text, meaning they look for what it could be expressing, assuming, valuing, or mean rather than what it does. This would also be demonstrated by a transition from centring of correctness and objectivity to situating it within a complex web of authorship and politics. These are not movements towards code becoming aesthetically pleasurable, but a change in attitude and interpretive orientation. Looking for meta-statements such as "I never thought of code in this way", changes in the vocabulary used to discuss code, and witnessing changes over time in where analysis is focused in the code text itself. We intentionally pluralised this indicator to give space for more than one poetic landscape to exist simultaneously. We acknowledge that participants may navigate multiple poetic ways of knowing, such as the earlier mentioned discovery of 'grace' in algorithms, while others may find it in the political subtext of the code. The audio recordings capturing comments and sentiment over time were used to assess whether this indicator was or was not produced. During the exit interviews, we were also able to more directly address this point through questioning, asking participants to reflect on whether they had experienced any changes in attitude toward code text (in regard to production, value, meaning, and content).

The second describes who is speaking in the group, at what frequency, and to what reaction from other participants. As mentioned earlier, one would expect technical experts to have sole authority to interpret code, however, if we shift the epistemic context in which we are operating then technical expertise is not the only dimension required to engage with the code text. Suddenly, for non-technical experts, code becomes legible without mastery and interpretation no longer depends on execution or correctness. What we would see is non-programmers speaking directly about the code text, rather than around it by referring to function or impact. Through the audio recordings, we assessed whether, over the course of the sessions, non-technical participants spoke up more in the group, speaking directly about what they interpret in the code text, and speaking with more confidence. Presence of this language is a sign of such an epistemic shift. Another indicator was whether disagreements and tensions do not immediately defer to technical experts to provide clarity or resolution and that technically apt participants are more receptive to the interpretations of the non-technical participants. From the field notes, we were also able to review the social tone of the session group; how was analysis accepted from non-technical persons? Was there tension? These non-linguistic indicators were gathered through the notes to further comment on how authority manifested over the course of the trials. The exit interviews also provided an opportunity to directly inquire about how individuals felt during the session. We were interested in whether non-technical participants felt more included or empowered to speak, if technical participants felt receptive to non-technical analysis, and how these sentiments may have changed from session 1 to session 3.

The third describes the kind of speech being employed in the group as

a whole and what speech is sustained for discussion rather than shutdown or avoided. Speculation without immediate attempts to collapse to ground-truth, tensions being sustained and explored rather than needing to be resolved, and mutually exclusive interpretations being permitted to coexist are all signs of a major epistemic shift in how we can understand code. The language was directly traceable from the audio recordings, and similar to the previous indicator, the field notes allowed us to comment on the group's reception and handling of this kind of language (if present).

The fourth demonstrates that there are poetic qualities latent in code that, when the mode of attention is shifted, perspective intentionality is adapted, and interpretive authority expanded, emerge congruently such that different groups repeatedly return to the same reappearing features. It speaks to the embedded aesthetic potential of code as a medium, the political density of code as an artefact, and expressive relationship authors can develop with their products. This was collected directly from the audio recording; listening to what points are made, categorising them, and tallying them to have an overview of poetic features and forms. We see the distinction between poetic features and poetic forms in that features are instantiative, for example in identifying one specific line or expression as possessing a quality like rhyme. However, forms are categorical groupings of these found features. For example, a singular metaphorical naming choice is a feature, while the broader category of 'semantic poetic forms' or 'syntactic word play' describes the relational strategy being employed. We draw this distinction to be better prepared to discuss a code-specific poetics, with describable techniques and characteristics.

4.4 Addressing Research Question

These indicators were designed to offer a guiding frame for analysis in order to address the research (sub)question. Indicators one through three address the main research question and indicator four, the sub-question. In the case of the latter, if we indeed collect a discretely categorised and tallied list of the poetic features commonly reappearing throughout the discussion, then we have indeed addressed this sub-question. The former is slightly more complicated in that we need to decide whether the method was 'effective' without having a comparable control. The presence of more indicators would signify more effectiveness. The presence of all three in a clear manner (so all participants demonstrate a clear change in perspective and self-report an explicit change in perception of code, non-technical report to feel more authority in interpretations and technical persons are receptive to this interpretive authority and both are reflected in the recordings and notes, and the language used does indeed transition from teleologically focused to speculative) would convey to us that this method is indeed effective at producing an epistemic shift from a teleological mode of attention to a poetic one. This does not mean the only or the most effective method, but simply that this is indeed a successful one; something which has yet to be explored and discovered.

However, with collective close reading being employed here as an exploratory rather than validated intervention, these indicators should not be understood as diagnostic measures of an internal epistemic state, nor does their presence alone establish causal efficacy. They function instead as sensitising criteria through which observable changes in interpretive practice can be traced. Importantly, the intervention itself encourages speculation, ambiguity and collective interpretation; the mere presence of such behaviours is therefore insufficient to demonstrate a shift. Greater evidential weight is placed on changes over time, persistence beyond direct facilitation, redistribution across participants, recurrence across texts, participant reflection, and moments in which alternative interpretations resist or transform an otherwise teleological return to technical explanation. Conversely, the absence, immediate collapse, or confinement of these behaviours would constitute evidence against the method's effectiveness under the conditions studied.

A failure to see any of these does not necessarily mean we cannot produce this reorientation, however if we see an active refusal to entertain these potentialities, and instead, see a noticeably contrastive reaction, this would suggest the method is ineffective. This would manifest as consistent and continuous defaulting to teleology to understand the code, a prioritisation of correctness and optimisation, and a shutting down of ambiguity (especially from non-technical persons). It would also look like almost all discussions collapsing to the single dominant voice of technical expertise, a trust and comfort in the hierarchy of technical mastery, and a convergence of interpretations to a singular accepted narrative. This would be challenging the collective aspect of the close reading specifically. Furthermore, if poetic language still appears but does not transform discourse in any meaningful way, that is also a sign that the poetic understanding is being superficially layered, rather than employed effectively. This would look like speculation, metaphors, and creative language, but no actual reflection on authorship, power, style, or subjectivity and no articulation of a feeling of epistemic reorientation.

Discovering a subset of them is still a sign of the potential for a reorientation of code epistemology through collective close readings. In the case of the presence of a subset of these indicators, a split group, or an ambiguous or unclear level to which an indicator is present, we were careful not to overestimate any specific interpretation of the data in signifying 'effectiveness'.

5 Results & Analysis

The following chapter presents the results of the collective close reading sessions and analyses the interpretive practices that emerged across the two groups. The study consisted of two groups, Group 1, comprising five participants: Charles, Laura, Marvin, Ada, and Alan; and Group 2, comprising of four participants: Claude, Gloria, Jacques, and Michael (the names of all participants have been changed in order to protect participants' anonymity). The groups were deliberately composed of participants from different disci-

plinary and technical backgrounds, including participants with formal experience in computer science or electrical engineering alongside participants whose backgrounds were primarily situated in the humanities (media or literary studies). Charles, Alan, and Ada, from Group 1, and Claude and Michael, from Group 2, were all of primarily technical disciplines. Laura from Group 1 and Gloria and Jacques from Group 2 were from humanities. Marvin was quite interdisciplinary. He had a background in art and programming. This composition provided a setting in which different forms of interpretive knowledge could encounter one another during the close reading of code.

The analysis is organised around the interpretive practices that emerged during these sessions. Rather than treating the participants' interpretation as isolated responses to the code, the analysis attends to how the interpretations were produced, sustained, challenged, redirected, and transformed through the collective discussions. The categories used throughout this chapter therefore describe recurring behaviours and norms of engagement identified, coded, and categorised across the transcripts of the sessions. They should not be understood as mutually exclusive; individual moments of discussion frequently intersect across several categories simultaneously. This is particularly evident in the later analysis of poetic features where technical, aesthetic, affective, linguistic, and contextual dimensions of the code often become inseparable.

The order of the analysis is intentional. We begin with the social conditions of the groups before examining the interpretive content that emerged from them. This is because the collective character of the method is not simply the setting in which interpretation takes place, it is part of what constitutes the interpretation itself. Who speaks first, whose observations are treated as credible, how readily participants challenge one another, and whether speculative or unfamiliar readings are allowed to persist, all influence what can subsequently be said about the code. The social dynamics therefore provide a necessary context for understanding the distribution of interpretive authority and the different forms of technical and non-technical engagement that follow. Beginning with these conditions also allows the later sections to distinguish between differences that arise from the code itself and differences that emerge from the particular social and disciplinary configurations of the groups.

Following this, the chapter examines interpretive authority, focusing on how technical expertise and disciplinary positioning shaped participation and how these forms of authority were negotiated during the sessions. The analysis then turns to non-teleological discourse, examining associative, speculative, and other interpretive movements that departed from questions of function and purpose. These address the first two criteria raised in relation to our research question. We then depart from the primary research question and address the sub-question, reflecting on the presence of poetic dimensions, first through the broader emergence of a union between technical and poetic interpretation and subsequently the specific poetic features participants identified within the code. The chapter concludes by returning

to the main research question and the final criterion, reconfigurations of understanding, as this is perhaps the most influential to the research question and most contingent on the rest of the content. In this final section we consider moments in which participants explicitly or implicitly described changes to how they understood code, poetics, and their relationship to one another.

Throughout the analysis, transcript excerpts have been lightly edited for readability. Filler words, false starts, and repeated words have been removed where their removal does not affect the meaning of the utterance. Omissions or larger portions are indicated by ellipses. Emphasis, pauses, laughter, and hesitation are retained where they are analytically relevant.

Each section will cover the discussed topic in detail. Quotes and behaviour will be shared and described, analysed and discussed. However, at the end of each section will a summary of the content and main conceptual points raised. Readers who wish to simply understand the main results and analytical points can read the final summary of each respective section and refer to the body for more granular detail.

5.1 Social Conditions

After completing both rounds of sessions with both groups, the influence that social dynamics and interpersonal cohesion played became clear. What was capable of germination within each group's trial depended heavily on the way in which the participants blended socially. Group 1 quickly developed a relaxed, light-hearted, informally-friendly relationship. This cultivated a more generous space, allowing for noticeably more open exchange and a higher rate of acceptance and exploration from technical participants of non-technical ideas.

> **Alan:** I'm waiting for *Laura* to be finished so I can correct her.
> *[Everyone laughs]*

The above quote from Alan, appearing about halfway through the second session, exemplifies the amiable atmosphere present in Group 1. The participant in this instance does not, in fact, offer a technical correction, instead adding to conversation about associations of the word *host* in the context of a server host. This moment demonstrates the social cohesion of the group; they are all comfortable with making jokes with one another *about* the technical disparity between them. It points out that Alan (and most likely all the other participants) are indeed aware not only of the experience disparity but how that has been affecting the dominating voices in the group. Alan seems to be making a reflective statement that this is something he has been doing over the previous sessions, but in a way that contributes to the dissolution of an interpretive hierarchy.

Group 2, on the other hand, seemed to remain suspended in some interstice between informal university students and formal experiment participants representing a domain. They were clearly more task-oriented with their remarks and lines of flight, often looking for immediate resolution of silences by the facilitator, obfuscating polemic remarks in diplomatic speech,

and closing conversations upon disagreement rather than exploring. They generally engendered an atmosphere which was noticeably more tense and frictional.

For example, upon reviewing the number of quotes marked from each participant, Claude's contributions were disproportionately represented in codes relating to technical closure, particularly moments where speculative interpretations were redirected toward singular technical readings.

> **Claude:** Yeah, I'm not very convinced that the discrepancy between this being a simple adder and the programme being a little bit more complex, I don't think that the difference between them is that deep to be honest. An adder is an adder, whether it's a programme or a logic gate, [...]. I don't think the programmers intended that gap to be there. Or think about the gap.

Here, Claude redirects the discussion away from the possible interpretive significance of the gap between implementation and form, toward functional equivalence; "an adder is an adder." The phrase collapses difference into technical sameness, treating the programme and the logic gate as interchangeable because they perform the same operation. This does not merely provide technical context, it attempts to limit the interpretive significance of the distinction being discussed.

In the coded[17] material, Group 1 generated 224 annotated excerpts, compared with 171 from Group 2; approximately 76.3% as many annotated remarks. This does not measure total speech or participation, but it does indicate that more of Group 1's discussion became analytically relevant to the categories used in this study. We see that comments that pertain to participants making an appeal to technical authority, minimising non-teleological readings, offering technical expertise, or defaulting to technical interpretation, are over represented in Group 2. This group produced 41 annotated quotes in these categories, in contrast to Group 1 producing only 39. Proportionally that is around 17% of Group 1's content being composed of this technical closure or bias, whereas around 24% of Group 2's comments. When we look at moments where participants explicitly left opposing statements to coexist (not feeling compelled to collapse to singular meaning) Group 1 produced 7 instances, whereas Group 2 only 3. To reiterate, these numbers are not reflective of total utterances. Furthermore, some of these quotes can share categories and differ in degree of severity, however it represents content we found analytically significant.

As we will see in later sections, Gloria and Jacques from Group 2 were also more polemic toward the discipline of computer science and engineering compared to Marvin and Laura. Gloria and Jacques tended to focus on political analysis, critiquing technological and political institutions and conventions whereas Group 1 usually remained at an aesthetic level.

[17] In this context, the term 'coded' is referring to the act of selecting a quote, annotating and categorising it with a *code* for analysis. This does not refer to software. Throughout this chapter the word code appears in both use cases.

Gloria: It brings this kind of...this tone of monitoring, I feel, as well. And then I'm just wondering with these words that are chosen, so collector, checker, recorder, inspector, I don't know. I feel that these are also words...they're not necessary to...sorry, I don't know how to say that.

Facilitator: Its okay; intentionality?

Gloria: There's intentionality between why they have chosen these words and they might not choose a word like monitor, maybe, or surveillance or something. It's like 'collected', [...] literally the same thing with 'cookies' as well. 'Cookies' is also an intentionally chosen kind of: "Oh, get the cookies." Like, "do you want the cookies?" Like, it's so that people feel it's more disarming. So I do feel things like recorder kind of has a multiple simplicity kind of understanding like it's not necessarily surveying per se. And a 'checker' is also a bit of an informal type of word, but 'inspector' then...that I don't know how I feel about 'inspector'.

Above is an example of such departure from structural or textual analysis. Gloria's interpretation moves from lexical observation to political critique. The names "collector," "checker," "recorder," and "inspector" are not treated as neutral technical descriptors, but as terms that modulate the tone of the software's activity. Her comparison with "cookies" makes this explicit: technical language is read as capable of softening, domesticating, or disarming processes that might otherwise be associated with monitoring or surveillance. Importantly, this critique emerges through close attention to naming rather than through an external political claim imposed onto the code.

These political readings did not simply add another interpretive layer, they also altered the interpersonal stakes of the discussion. Technical participants may have experienced critiques of computer science culture as critiques of their own disciplinary position, perhaps leading to Group 2's conversations becoming more readily defensive or divisive.

However, this tension also made Group 2 especially revealing for the study, because it exposed the conditions under which technical authority, disciplinary defensiveness, and political critique shaped the limits of poetic engagement. It is clear that there is more contributing to the precipitation of poetic meaning than simply content and background. The idiosyncratic composition of the groups themselves was especially conducive. Group atmosphere was not incidental background context; it shaped the degree to which participants felt able to speculate, disagree, resist technical closure, and sustain non-teleological interpretations. It shaped the participants' willingness to take interpretive risks.

One potential contributing cause may have been the circumstances under which the groups formed. Group 1 was composed of students who

responded to an open call[18].Each participant was selected by on the basis of their self-described backgrounds. Preference was given to people who seemed embedded in their background with active attention to avoid highly interdisciplinary participants. The second largest factor for consideration was availability. Interestingly, despite screening for mono-disciplinary backgrounds, it seems everyone had at least some minor exposure to their alternative discipline or, at the very least, an interest in it. For example, in session one, all participants of Group 1 described having had a period of reading (and some writing) poetry for personal pleasure.

In session 2, prior to the beginning of the session, as we waited for the last few stragglers to arrive, Alan and Laura began to share poetry from their respective home countries with those of us already present. This was unprompted and occurring even before the facilitator's arrival. They began talking about the practice of reading and translating from their own mother-tongues to the group's shared language, English. We listened to them share poems first in their own native languages, and then live translated in English. Alan, after the end of the third session, created a group chat for Group 1, as well as others invited, to continue sharing poetry and having discussions about the nature of poetic reading and interpretation. He is still encouraging all to organise another meeting.

In contrast, Group 2 was formed entirely via invitation from the facilitator. Each participant was chosen for their mono-disciplinary background. This means they did not express explicit interest in the exercise before being invited unlike all the participants of Group 1. This may have directly translated to the lack of openness and willingness to engage in exercises that were unconventional or seemingly unorthodox. Also, without this shared in interest in the activity, we see less of a social foundation within the group. Unlike Group 1, Group 2 did not share pleasantries, socially interact, or engage each other outside the context of sessions.

These differences in group formation and informal sociality suggest that collective close reading cannot be understood as a method applied uniformly to any collection of participants. The method depends not only on the code excerpts, the prompts, or the disciplinary backgrounds present, but on the social atmosphere through which participants become willing to risk interpretation. Group 1's cohesion made speculative and non-teleological readings easier to offer, extend, and sustain. Group 2's tension, by contrast, made visible the points at which technical authority, disciplinary defensiveness, and political critique could restrict or redirect poetic engagement. In this sense, social cohesion was not peripheral to the analysis; it formed part of the conditions under which poetic meaning could, or could not, precipitate.

[18]The open call was shared among the students of LIACS participating in the class AI in Society. This class was mostly populated by students studying Creative Intelligence and Technology (M.Sc.) and Artificial Intelligence (M.Sc.), though as an elective it was open to students from all faculties at Leiden University. They were encouraged to share it with other, external relevant parties.

5.2 Interpretive Authority

When we refer to *interpretive authority*, we are describing the ways participants implicitly or explicitly negotiated who was entitled to explain, validate, redirect, or close down interpretations of code. Across both groups, technical authority was immediately present, but it did not operate as a simple hierarchy in which technical participants always dominated and non-technical participants merely deferred. Rather, authority appeared through conversational timing, disciplinary confidence, requests for explanation, moments of resistance, and the shifting use of technical knowledge itself. Collective close reading did not remove technical authority; it made that authority visible, contestable, and, at times, available as a resource for poetic interpretation.

5.2.1 Interpretive Asymmetries

An asymmetry in immediate authority distribution was noticeable. Technical authority appeared first not through explicit claims of expertise, but through conversational timing. You could see that immediate responses were usually technically oriented and while this does not mean they necessarily came only from technical participants, the first associations and attempts at interpretation were usually technical. With that being said, technical participants were indeed quicker to respond. This is especially true for Group 2 where Claude and Michael really dominated the conversations in session 1. Technical participants tended to respond more quickly, more confidently, and often setting the initial interpretive frame.

If we look at the very first response to the introduction of the code in session 1 for Group 2:

> **Michael:** I mean coming from electrical engineering, I can recognize this.
>
> **Claude:** But the thing above as well?
>
> **Michael:** Well, the circuit I recognize these are clearly logic gates.

Following this immediate response, of the 129 transcribed utterances, 62% of what followed was from Michael and Claude, mostly in dialogue with one another as they explained or investigate the functional purpose of the code. Only 19% were from Gloria and Jacques, most of that was from Gloria asking to explain certain terms or concepts brought up during Michael and Claude's discussion. The remaining comments were from the facilitator answering questions. These values represent raw instances, and does not capture the length of a participant's comments. Claude and Michael would often delivering long monologues that also consumed much of the discussion time.

> **Michael:** So that [...] that's perfectly fair. So all of these lines we can think of them as just like wires. And these are our inputs for the circuit. In here we can put a zero or a one, right?

This is for working with binary. And the idea appears to be...So from here, you can follow this line going into here, and it looks like it also branches off into this one, right? I don't know how well you can see. And this one goes into the input here.

Jacques: Um I'm...I'm...I'm taking guess. *[struggling to see Michael's gesturing]*

Michael: Yeah. So from here the signal goes in and it makes its way into here. Yeah. And it also follows and it makes its way into here.Then this one does the same with this input and it also goes into the other input here, right? Then once it reaches each of these Logic gates, it'll do an operation. Let's talk about this one first. This is an AND gate. What it will do is it'll look at its inputs. So the inputs, each input can only be one or zero. If it receives two zeros, it'll output a zero. If it receives a one and a zero, it doesn't matter which one's which, it'll output a zero. And if it receives two ones, it'll output a one. So it'll only ever output a one if both of its inputs are zero. I mean if both of its inputs are one. So that's an AND gate, right? Because it's evaluating input A and input B. If both of those are one, it'll output a one. Then we have this gate here. This is an OR gate. So what it'll do is it'll once again check both of its inputs And if either of the two inputs is a one, it'll output a one. It will also output a one if both inputs are one. So if it receives 0-1, it receives it outputs 1. 1-0 outputs 1.

As previously discussed, the proportion of unprompted technical expertise offerings and minimisations was disproportionate on behalf of Group 2. We see similar trends when looking at the distribution of comments within the trials, across sessions. Group 1 still offered technical explanations throughout the trial, however we see out of the 14 annotated comments in this category of offering unprompted technical expertise, 43% of them were in session 1, with a steady decrease over sessions 2 and 3 (35% and 21%, respectively). It also decreases within each session, with the majority of technical expertise being provided toward the beginning, and lessening as the group discusses. Group 2 demonstrated a different pattern. There annotations were uniform across all the sessions (3 annotations per session). However, they did also tend to decrease within sessions, similar to Group 1.

In both groups, we saw also a tendency for technical information to become a means of providing substance, depth, and contextual richness to interpretations. What we saw is that over the sessions, the technical perspectives tended to be offered in service of the groups' collective interpretation whether technical or not. So, technical information was being provided as context or even inspiration for poetic or alternative readings. A clear example of technical knowledge becoming interpretive support appears in Group 2, when Michael explains the function of return statements while simultaneously translating them into a poetic structure.

Michael: I think one thing that we'll see as we see other func-

> tions and methods, is that a lot of them will end, big structural repetition things, will end with a return statement. So from a structural kind of repetition, if we think of a function maybe as like one one block of prose or of verses. The 'return something' could be be like a kind of a larger meta repetition element that is found across the entire piece.

We see here that Michael explains why this word return keeps returning, but does so in a way to compose a literary analogy that can perhaps construct a skeleton for poetic interpretation. The return statement is first explained functionally, then re-read as a potentially poetic device emerging from the repeated structural occurrence. Later there will be a section dedicated to the union of poetic and technical features, but here we see it organically arise from a technical participant without prompt. In both groups however, the offering of technical perspectives for the purpose of collapsing interpretive meanings never fully disappears.

5.2.2 Definitive Authoritative Figures

In both groups, technical authority did not remain evenly distributed among all technically experienced participants. Instead, each group developed a participant who became especially associated with technical legitimacy. In Group 1, Alan often occupied this position, though his authority softened over the course of the sessions as he became more willing to accommodate speculative and non-technical readings. In Group 2, Claude occupied a similar position more rigidly. Although he expressed in the exit interview that he valued the diversity of interpretations, the sessions themselves show more frequent moments of defensiveness and technical closure. This contrast suggests that technical authority was not simply a matter of disciplinary background, but of how particular participants inhabited and performed that background within the group.

> **Alan:** One thing I wanted to mention [...] to complement *Laura* and *Charles'* points is that this could storified. Instead of saying "okay, *private cookieRecorder* is actually *private cookieChecker*" or something, and the ones are like "*private beaconChecker*". So, later on, you can use the syntax and semantics to say: "okay, cookieRecorder, go check cookies!" instead of saying just, I don't know. 'inspect cookies'.
>
> **Laura:** You! You're old enough now, you can leave the house, you can do the job.
>
> **Alan:** Yeah! You go on and inspect cookies, you go on and inspect beacons, you, you do this. [...] You as the reader or the programmer, you're telling these, "okay, you go out there you do these inspections and report back to me."

Here, Alan does not use technical knowledge to restrict interpretation. Instead, he draws on syntax, semantics, naming conventions, and object-oriented structure to extend the group's narrative reading of the code. His

suggestion that the code "could be storified" indicates a shift from technical correction toward technical-poetic elaboration.

> **Claude:** I just wanted to say, because I can talk about this for hours, programmers' language (like spoken language, I feel), there are lots of specific field terms. Like *[Gloria]* said, the cookies that seem very weird to outsiders but not necessarily for programmers and I do feel like it's very important to highlight because it's just a whole different field. [...] I do feel like the contentions are very different from what it looks like from an outside perspective.

Claude's response does not directly reject the political or poetic interpretation of 'cookies,' but it reframes the disagreement as a problem of disciplinary perspective. By emphasising programmer language as "a whole different field," he reasserts the relevance of insider knowledge. This is not technical closure in its strongest form, but it does mark a boundary around interpretation. The outsider's reading is made provisional because it lacks access to the connotations internal to programming culture, centring the technical programmer and their perspectives. For context, this quote came in the second half of the third session, so at the end of the trial, Claude was still operating from a position that orbits a technical anchor. Alan's comment was made at the very end of the second session.

This comparison between Claude and Alan is not to highlight personality differences, though they most likely influenced these notable interactions, but instead to reaffirm that authority over interpretation and the positioning of technical participants was indeed a social dynamic as much as a disciplinary one.

Despite the growth in openness from Alan, moments of minimisation toward non-technical readings of the code text still predominantly came from him. However, many of the minimisations were paired with a self-reflective comment about computer science and engineering naturalisation that prevented him from being to engage with the alternative perspective, not that he thought it wasn't there. This sentiment is evident in Alan's comment:

> **Alan:** In some sense it's harder for me to get extraordinary meaning or creative story out of [the code text] because I know the syntax and semantics of it. It limits my creativity in terms of story and stuff. Like I saw *Laura* [...] connecting host to party or some other words but I just couldn't because my brain is hardwired in this case that host is this.

These kinds of comments rarely came from Claude or Michael. They more commonly engaged in the direct closure or minimisation of speculative interpretation, usually making the claim that Gloria and Jacques lack context and background knowledge (this appears again later in the discussion of sustaining speculative conversations). The contrast between Alan and Claude shows that technical authority was not reducible to disciplinary

background. It depended on how a participant performed that background within the group: as a reflexive and generative resource in Alan's case, or more often as a stabilising disciplinary boundary in Claude's.

5.2.3 Non-Technical Resistance

Despite the hitherto discussion over dominant technical voices, we must be clear that the alternative positions were not merely represented, but pushed and defended throughout the sessions of both groups. There are numerous examples of immediate resistance to attempts of interpretive closure. This could be because of the framing of the sessions in general as interdisciplinary and therefore, regardless of not coming from a technical background, the humanities participants expected to be listened to.

In Group 1, all of these moments of *'pushing back'* against normative technicality come from Laura, the primary humanities participant. She was most defensive in session 1, but became less defensive over the next two sessions, most likely due to a combination of an increased social cohesion of the group and the technical participants simultaneously became less rigid in their technicality.

> **Laura:** But it's functional as well as form. But if the person who wrote it said, "okay, this is to soothe the computer in a way," then it might just be that the computer is being considered as something that has a relation of some kind; that the computer has a different underlying layer of meaning towards that that could make it feel uncomfortable with something that it doesn't know. So it does attribute to the computer something.
>
> **Charles:** I think they were being cheeky
>
> **Laura:** Yeah, but still, if we look at it, if we take it as actual [...] standpoint that they would defend or whatever, but if that's what's said...because, I don't know, it's not particularly *aesthetic* in a way if you think about poetry in an *aesthetic* way.

This exchange shows Laura resisting an attempt at interpretive minimisation. Charles' comment reframes the author's statement as humour, which risks closing the speculative reading by treating it as merely playful. Laura does not deny the humour, but insists that the premise can still be analysed. Her response keeps open the possibility that the joke itself attributes relation, discomfort, or interiority to the computer. In this sense, she sustains a poetic reading by refusing to let playfulness become a reason for interpretive closure.

We do see this behaviour in Group 2, but with a different materialisation. The first session has only a few moments of non-technical participants pushing their alternative perspectives, however the ones present are quite strong.

> **Michael:** I mean I think one thing to consider: it might partially be a practical consideration A 16-bit adder would take up

> a lot more space. But they could have picked a smaller word, maybe something with two characters. But then, I think you lose some aesthetic quality by having too many different things [...] because as people, we don't necessarily distinguish between things like in case sensitive ways. Capital M is M, same way lowercase M is of course, it has significance in written text, the capitalization.
>
> **Gloria:** I don't know, I do think that there is. Obviously, there is a very significant difference, also just like an emotional reaction to like a capital M for people, a capital M and a lowercase M. Because after the fact that it has a function, if something operates with this function and has a meaning attached to that for so long, that meaning is not just coming from the function, it becomes kind of like interchange, not interchangeable but coexisting, one kind of informs the other. So a capital M for me as a human might mean something a lot more aggressive, even though a capital M in language function doesn't necessarily mean aggressive specifically; but it is used to signify shouting, for example.

This exchange demonstrates that resistance to technical authority did not necessarily take the form of rejecting technical explanation. Michael's comment offers a practical and perceptual account of character choice, considering space, visual similarity, and case sensitivity. This contextualisation also orbited a discussion of the programming language C's handling of variable names, white space, and case sensitivity. Gloria's response does not deny this technical account, but refuses to let it exhaust the meaning of the text. Her emphasis on the affective difference between uppercase and lowercase letters reframes the technical distinction as also cultural, embodied, and emotional. In doing so, she articulates one of the central tensions of the study: once a technical form circulates as readable text, its function and its meanings begin to coexist and inform one another.

The second session of Group 2 had less of these, and most of these interactions occurred in the third session. We hypothesis that this may have arisen from a social exhaustion; with Group 2 having a less pleasant social atmosphere, by the end of the trial the humanities participants were more prone to confrontational and polemic exchanges, unlike Laura who perhaps in experiencing a more friendly group, became more tolerant as a result. All participants bar Claude engaged in pushing back against some technical interpretation further demonstrating Claude's role as a sort of *software engineering monolith.* Even Michael, an engineering peer was pushing heavily to consider other ideas at times. Michael was noticeably more open to alternative ideas, this accounts for the reduction of confrontation in session 2 when Claude actually had to be absent. We see that this session was most fruitful in speculative conversations throughout the categorical analysis.

5.2.4 Appeals to (Lack of) Technical Experience

Participants (usually non-technical but also technical at times) referred to a lack of technical expertise as a reason for them not wanting to engage in *any* kind of interpretation, and other moments where they explicitly asked for technical information (either from the facilitator or peers).

However there were differences in how these moments manifested between groups. In Group 1, Laura often used these moments to motivate why she was offering a non-technical interpretation.

> **Laura:** The type of thinking, the logical construction of those types of concepts, I think deep down I just don't care. I get lost. I was listening [to the group], but I still don't really understand it and for me it's just a very visual thing of looking at stuff that I'm familiar with, not because I know the concept but because I see it and I circled all the capital 'main's and the [lowercase 'main's], because there's a symmetry. [The technical meaning] doesn't hold a specific meaning where I have the feeling that I can discuss about because I don't have an opinion.

Other participants would offer technical explanations to these comments, sometimes Laura would accept them, but more often they would be used to contextualise or deepen her own non-technical reading of the code. Often, these comments would open up the floor of non-teleological or speculative discourse. For example, in the instance above, the conversation prior to Laura's comment revolved around the meaning of *main* in a coding context and the substitution strategy employed in *Adder* (source code can be found in 4.2). Afterwards however, the conversation departed and began a discussion on the nature of satire and comparisons to *concrete poetry*[19].

There was only one annotated comment of this nature in Group 2 (coming from Gloria in the first session). I suspect this is because the technical/non-technical division was much clearer. In the first session, conversations usually began as highly technical and were dominated by Michael and Claude, this meant Jacques and Gloria did not ever have to clarify their non-technicality, it was implicitly set up by their lack of participation.

The most rare phenomenon was appealing to technical authority. This was unexpected. We anticipated technical participants to use this as a means to push their readings onto others but it did not happen very frequently. It happened only twice in session 1 for Group 2 and was more to explain why they had immediate knowledge of what they were looking at, not necessarily to shut down any conversation. The more common form of this phenomenon, something which appeared in both groups, was a suggestion that having a technical background was necessary to truly appreciate the poetic qualities of the code. This sentiment was not shared by the non-technical participants and the technical participants accepted that it was not necessary after discussion.

[19] Concrete poetry is poetry wherein the text is arranged on the page such that its shape becomes a visual analogy for the subject matter of the poem. The words of the poem illustrate the content of the poem. It is sometimes referred to as *shape poetry* (Hardin, 2024).

> **Marvin:** Maybe to find the poetics in code you need to be a programmer. You need to be the same way as like I can really appreciate a song in a language I don't understand, but reading a poem in a language I don't understand becomes a lot harder to appreciate.
>
> **Laura:** [...] [Alan] read his poem in his language. And it was like '[he'd] say like two verses, whatever, ten lines, and then we asked him, "okay, but what does it mean?" And then he said like three lines. And I was like, that can't be all of it. "Yeah, but that was the gist of it."
>
> **Marvin:** So I think there's some there.
>
> **Laura:** I disagree. I disagree on the poetic stuff, that you need to be a coder for seeing something poetic in it.
>
> **Marvin:** I'm not also not that set on it.

And later, in Group 2:

> **Claude:** You have to have some fundamental understanding of...for poetry you need to have a fundamental understanding of how the language works. And for code, for stuff like this, you also need to have fundamental understanding of both programming and I guess electrical engineering.

Taken together, these exchanges suggest that interpretive authority in the sessions was neither abolished nor simply reproduced. Technical knowledge continued to matter, shaping who spoke first, who explained, and whose readings were treated as immediately plausible. Yet technical authority rarely appeared as an explicit claim to superiority. More often, it operated through subtler conversational forms such as the swiftness of responses, disciplinary contextualisation, correction, self-positioning, and the framing of non-technical readings as either legitimate interpretations or insufficiently informed impressions. Collective close reading made these dynamics available for negotiation. It allowed technical knowledge to become a scaffold for poetic interpretation, while also creating moments in which non-technical participants could resist the idea that technical fluency was the condition of meaningful engagement with code.

5.2.5 Summary

Across both groups, interpretive authority initially tended to favour technically experienced participants. They were usually quicker to respond to the code, and their responses more often established the first frame through which it was discussed. This was especially pronounced in Group 2: in the first session, Michael and Claude accounted for 62% of the 129 transcribed utterances, while Gloria and Jacques accounted for 19%, much of which consisted of requests for technical clarification. Group 1 demonstrated the same tendency less strongly. Unprompted technical explanations also became less frequent as discussions progressed, particularly in Group 1, where

43% of the annotated instances occurred in session 1, falling to 35% and 21% in sessions 2 and 3 respectively.

Technical authority was not distributed evenly among technically experienced participants. Alan emerged as the clearest technical authority in Group 1, while Claude occupied this position in Group 2. Their use of that authority differed considerably. Alan increasingly reflected on the limitations imposed by his own technical familiarity and began using technical concepts to extend poetic readings; Claude more consistently used disciplinary context to qualify, redirect, or delimit interpretations offered by non-technical participants. Technical expertise therefore did not determine a single interpretive behaviour: its effects depended partly on how individual participants performed and negotiated that expertise within the group.

Non-technical participants were nevertheless not passive recipients of these hierarchies. Laura repeatedly resisted technical minimisation in Group 1, while Gloria and Jacques increasingly defended affective, cultural, and speculative readings in Group 2. These disagreements rarely involved rejecting technical explanations outright. More often, participants accepted the technical account while disputing its sufficiency. For example, Gloria accepted the functional significance of case sensitivity while insisting that uppercase and lowercase letters simultaneously carried affective and cultural associations. Participants also occasionally appealed to their own lack of technical knowledge, although this could itself become generative: Laura, for instance, used her inability to follow a technical discussion of main to redirect attention toward visual symmetry, eventually opening discussion of satire and concrete poetry.

Overall, collective close reading did not abolish technical authority, nor did it produce an equal distribution of interpretive power. Technical knowledge continued to influence who spoke first, who explained the code, and which interpretations initially appeared legitimate. What changed was how that authority could be used and challenged. Across both groups, technical explanation increasingly became material for poetic interpretation as well as a means of resolving uncertainty, while non-technical participants repeatedly resisted the assumption that technical fluency was a prerequisite for meaningful engagement. Interpretive authority therefore remained uneven, but became more visible, negotiable, and capable of serving multiple forms of understanding.

5.3 Speculative and Non-teleological Discussion

When referring to non-teleological content, we are addressing any theme of conversation that discusses dimensions not pertaining to function, purpose, or execution of the code. Speculation often appeared within these discussions, but speculation and non-teleology are not identical. Participants could, and did, speculate about teleological matters. However, in this context, speculative remarks are primarily referring to moments when the groups were discussing things that could not be confirmed by any imme-

diately accessible technical context, such as the author's intentions, worldview, or imagined audience. The most frequent form of non-teleological discussion was when groups considered and examined poetic instances within the code. These have been designated to a chapter of their own both for thematic separation as well as due to the volume of these recorded moments. This chapter focuses on cultural and media associations generated between the code texts and readers' and speculations into intentions and character of the author

5.3.1 Association Building

Following poetic instances, the next most frequently appearing non-functional theme was about inspired associations. This was remarkably popular among Group 1. Associations refers to a diverse range of topics from visual associations, comparisons to traditional poetry genres/types, acoustic and rhythmic associations, and broader associations with media, (pop) culture, and more.

Both groups used association as a means of making sense of the code from the first session onward. In many discussions, associative responses were among the first interpretations offered. This category overlaps heavily with the later discussion of poetic dimensions, particularly visual aesthetics, acoustic affordances, rhythm, and broader commentary. However, the emphasis here is not on the poetic feature itself, but on the movement of thought produced by association; the way participants departed from immediate function and allowed the code to gather relations beyond its technical operation.

One unexpected finding was that the second code text generated more associative connections than the first across both groups. Prior to the sessions, we anticipated that the first code text, due to its obfuscated and explicitly aesthetic character, would more readily invite associative interpretation. Instead, the fewest associations were generated in the first session of both groups, while the second session, in which the more conventionally structured code text was introduced, produced the greatest number of associative comments. Analytically, we see that of the 53 total associative comments annotated during our research, approximately 26.4% arose, from code text 1, Adder (22.2% of Group 1's comments, and 35.3% of Group 2's). Correspondingly, 73.6% of the associative remarks, then, came from the second code text, class Inspector of the EU's Website Evidence Collector (77.8% of Group 1's associative comments, and 64.7% of Group 2's). This suggests that associative reading was not simply produced by obvious aesthetic strangeness. Rather, the more recognisably functional code text may have offered participants more points of friction between ordinary technical structure and interpretive possibility.

Participation in associative reading was also distributed differently between groups. In Group 1, all participants engaged in some form of association-building, although Alan and Ada offered comparatively little. This absence is itself informative. Alan explained that his engineering training made it more difficult for him to perceive alternative associations quickly, describ-

ing a kind of disciplinary conditioning that oriented him toward technical meaning before poetic or associative meaning. However, in the exit interview, he also clarified that he did not consider such associations useless; rather, they appeared to him more slowly, helped by the frequency with with his interpretive colleagues were producing novel readings.

> **Alan:** Whenever somebody else like *Laura*, somebody else who had no programming experience, especially with the programming language, it made me think: what made this person think of this line of code, that I like thought was so basic, why did she or he interpret it this way? What was the reason behind it? So, it made me think *more* about it, and like, think of this *affects* that were on it.

Alan, had 9 overall comments where he related the code text to something else, examples include associations to ASCII art, German poetry, enriching uranium, to name a few. In the first session, he produced only one instance, in the last session, he produced 5. In this sense, his limited participation in association-building does not indicate rejection so much as delayed access.

In Group 2, associative reading came primarily from Gloria and Jacques, the non-technical participants. Michael also offered associations, though these were often blended with technical contextualisation. Claude, by contrast, did not produce any annotated associations, whether blended or otherwise. Instead, he often responded to associative readings by explaining them through technical convention, historical programming culture, or missing contextual knowledge. Michael occasionally engaged in similar explanatory moves, though to a lesser extent and usually with more openness to the alternative interpretation being proposed.

The contrast between the groups therefore suggests that association was not merely an individual habit of interpretation, but a socially conditioned practice. Group 1 produced more than twice as many associative comments as Group 2, and this difference cannot be explained by the code texts alone, since both groups encountered the same materials. Rather, association appears to have depended on the degree to which participants felt permitted to make interpretive leaps without immediate correction. Where the social atmosphere was more relaxed, associations proliferated. Where the technically grounded voice of the group was stronger, associations became more politically charged, more contested, or more likely to be redirected toward technical explanation. Association therefore marks one of the first ways participants loosened the code from immediate function, but it also reveals how unevenly such loosening was permitted across the two groups.

5.3.2 Killing the Author

The next most common form of speculative, non-teleological discourse concerned the figure of the author. This includes aspects such as their persona, profile, intentions, world-view, humour, priorities, or imagined relation to an audience. This category was closely linked to many others, as discussions

of visual form, naming decisions, technical implementation, layout, and audience often led participants to ask, implicitly or explicitly, why the code had been written in this way. Authorial speculation therefore functioned as one of the primary ways participants moved from textual observation to interpretive projection.

This form of discussion is important because it required participants to remain with claims that could not be confirmed through the code text alone. To dwell among speculative claims. Unlike a functional explanation, which could potentially be verified through technical context, authorial speculation remained necessarily open. Participants could not know with certainty what the author intended, what values shaped their decisions, or what kind of audience they imagined. Nevertheless, these questions repeatedly became productive sites of interpretation. In this sense, authorial speculation became one of the clearest examples of sustained non-teleological discourse.

These discussions often appeared when a particular feature of the code seemed to exceed immediate functional necessity. A visual arrangement, a naming decision, a repeated structure, or an unusual technical choice would prompt participants to ask whether the feature was deliberate, playful, communicative, satirical, or ideological. Participants used the code fragments to construct possible authorial positions. In the case of Adder, we saw a playful programmer. In the Inspector class, a bureaucratic institutional actor. At times we saw a technically constrained engineer, others, a satirisation of software convention.

As with associative readings, the character of authorial speculation differed between groups. Group 1 tended to approach authorial intention through play, visuality, aesthetics, and audience engagement, particularly with the first code text. Their discussions often revolved around whether the author was joking, performing, inviting confusion, or constructing a visual or textual experience for the reader. Group 2, by contrast, more frequently politicised the authorial position. Their authorial speculation moved toward questions of institutional voice, technological culture, surveillance, and the values embedded in naming and classification. The 'author' of the second code text was therefore often treated less as an individual programmer than as a situated techno-institutional perspective.

> **Alan:** There were a lot of Persian poets that, you know, somebody came up to them. [And] there are a lot of different stories of different poets that had this somebody came up to them that "Okay. If you're a good poet, write a poem with these words in this tone and this meaning!" and they would write it or sing it right away, just because they were *that* good. I think [the authors who made Adder] because they said, "Okay. I did it because I could...and it works, yeah. And it's because I'm better than y'all."

Here, Alan does not treat the code's constraint as merely technical difficulty. He reads it through a tradition of poetic virtuosity, where skill is demonstrated precisely by accepting arbitrary formal limitations. The

imagined author becomes a performer of capability, someone who writes the code not only because it is useful, but because its successful execution under constraint becomes a gesture of aesthetic and technical superiority.

> **Jacques:** I guess it could be also if you are trying to understand the programmer's perspective, like rather than use using 'update' or something 'enrich' also could function in a way to say that this data that was collected is important and it's enriching the data set for example.
>
> [...]
>
> **Jacques:** Yeah, so it's like kind of like a psychoanalytical [or] psychoanalysis of the programmer. We could read it that way...

Jacques' reading treats the word choice of the programmer not as a neutral technical descriptor, but as a trace of evaluative orientation. To "enrich" a data set is not merely to alter it, but to improve it, increase its value, or make it more complete. In this moment, authorial speculation becomes a way of reading value judgments embedded in technical language. The "programmer" is not imagined playfully, as in Group 1, but as a subject whose terminology reveals a relation to data, collection, and institutional value.

In Group 1, over half of the annotated comments concerning authorial intention occurred in the first session. In Group 2, the highest concentration appeared in the second session, when Claude was absent, though these discussions still largely circled back to the first code text. This distribution suggests that authorial speculation was shaped both by the affordances of the code and by the social conditions of the group. The more explicitly aesthetic code text invited participants to imagine an author more readily, while the absence or softening of technical closure made such speculation easier to sustain.

Participation was also unevenly distributed. In Group 1, all participants engaged in discussions of authorial intention, suggesting that the social cohesion of the group allowed speculation about the author to become a shared practice rather than a disciplinary niche. In Group 2, these discussions came primarily from Jacques and Gloria, with Michael occasionally contributing through technically contextualised reflection. Claude engaged far less in authorial speculation, and when he did, his comments more often redirecting the discussion toward technical convention, missing context, or programmer culture. This again reflects the broader disciplinary distinction between the groups: authorial speculation flourished where participants felt able to treat the code as an expressive artefact rather than as a technical object whose meaning was exhausted by context or convention.

Authorial speculation therefore occupied a central place in the emergence of non-teleological discourse. It allowed participants to treat code not only as a sequence of instructions, but as the residue of decisions, attitudes, values, jokes, constraints, and imagined audiences. Importantly, these discussions did not require certainty. Their significance lies precisely

in the fact that participants were willing to dwell in interpretive possibility without demanding immediate verification. Through speculation about the author, code became not merely functional artefact, but an authored and situated text.

The final dimensions of non-teleological discourse concerns not simply the appearance of speculative remarks, but the extent to which speculative conversations were waded through, dwelled within, sustained. By this, we are referring to moments in which participants collectively remained with an unconfirmable potentiality, something that could not be immediately explained with technical information. This differs from *association*, in that personal associations, for example a visual association, can often be defended through an appeal to subjective perception: *"this is simply what I see and therefore is not deniable nor collapsible!"* While such claims indeed appeared, we discuss many of them in depth, they are speculative without requiring extended justification or collective agreement. Instead, if we take claims regarding the intentions of an author, their imagined audience, the values embedded in their naming decisions, the revelations about computer-science culture exposed by code, or revelations about the European Union, surveillance, or beauty; these all demand a certain interpretive tolerance from the group. They ask the group to dwell in the unconfirmable without seeking refuge in that which is merely private.

Authorial intention was the most common form of sustained speculation, as discussed. Other frequently appearing forms were in discussions of audience (and nature of audience and gaze itself), institutional context, cultural meanings, and broader commentary. Participants asked not only what the code did, but what kind of relation it established with a reader, what kind of world it seemed to assume, and what values or habits of thought might be crystallised. These conversations often exceeded what could be known directly from the fragment being discussed itself, yet they remained anchored to that fragment, not becoming arbitrary.

Speculative conversations appeared to be received well by most participants. In exit interviews, all technically trained participants described these discussions as at least enjoyable, and in some cases illuminating. Even Claude, who most often redirected speculative readings toward technical explanations, remarked that several of the conversations would remain with him after sessions.

> **Claude:** It mostly made me feel interested in what *[Gloria and Jacques]* thought they had to say about the subject. Or what they thought was, well, correct about what's in the *'poem'* or the code itself, yeah. But there were also parts that they, at some point, they pointed some things out and I was like, oh yeah, that's true! I'll remember that!

We saw similar sentiments from Alan who also expressed that it was something that developed overtime, with exposure to the practice, rather than it being a position held a priori.

> **Alan:** I think it was something that came in small steps. Like,

> the first time that it happened I was like: "okay, it...it's maybe because they don't know about *cookies* in general and..." but, after a while I was like: "yeah, they're...they're on to something, like it's kind of making sense." Like *private cooking inspector* and stuff. These stuff are somewhat, like, sticking with me."

This matters because it suggests that speculative discourse was not simply tolerated by non-technical participants while technical participants waited for a return to "reality". The participants most attached to technical framing could still recognise value in conversation that moved beyond verification.

Speculative conversations were most common around the first code text. This is unsurprising since in its obfuscated nature, the context of its production as a contest entry, and the explicit visual considerations, invite questions about intention, performance, and audience. Across the sessions, however, these conversations taper. This was especially noticeable in Group 2, where Claude became increasingly quick to explain interpretive openings through technical convention or teleological necessity (contrary to his claims of openness to speculative practices in the exit interview). It is also worth considering that when we count the number of speculative conversations, it does not capture the length of time these conversations were sustained. In many cases participants spoke at length for extended periods, and while participants like Claude or Alan may have offered closures, they were sometimes ignored. This produces a slight bias in measuring closures versus endurances.

These aforementioned closures also manifested differently per group. In Group 1, it came primarily from Alan and Charles at the same rate (each producing 7 instances of a default to technical explanation). However, in Alan's case, it is worth noting that despite his position as the participant most associated with technical knowledge, his technical explanations were often offered as contextual support rather than as attempts to end discussion. Whereas with Charles' contributions, they appeared more often in moments where speculative lines of thought were redirected or minimised.

In Group 2, by contrast, closure came most frequently from Claude, whose responses often returned the group to technical context, programming convention, or the insufficiency of available background knowledge. However, closure was not exclusively a technical practice. Michael, Jacques, and Gloria also occasionally shut down particular speculative lines, sometimes from non-technical positions. This suggests that closure was not only a matter of disciplinary hierarchy, but also of social receptivity. Group 2's weaker cohesion made participants less willing to extend one another's interpretive possibilities, even when those possibilities did not come from technical authority. If we look at the distribution over the sessions, out of the 17 moments where a speculative conversation was minimised by technical explanation, 0% appeared in session 2 where Claude was absent. 8 appeared in the first session and 9 in the last, where he was present. It is clear that despite Claude's later comments about speculative acceptance, his behaviour created a different atmosphere in practice.

Another important indicator for this research was about the facilitation of co-existing interpretations. Upon analysis, we tended to see the alternative case, a convergence to singular interpretations. Participants were generally willing to explore uncertain claims, but they usually did so by trying to align around a shared interpretation. The norm was not a proliferation of equally sustained possibilities, but a movement toward a single line of thought that the group could collectively inhabit. This is important because it qualifies the openness of the method. Collective close reading did not automatically produce interpretive plurality in the strongest sense. It produced conditions in which non-functional interpretations could emerge and be sustained, but those interpretations still often moved toward consensus. The value of the method, then, was not that it dissolved the desire for closure altogether, but that it temporarily delayed closure long enough for other kinds of meaning to become available.

5.3.3 Summary

Across both groups, speculative and non-teleological discussion most often emerged through association-building and speculation about authorship. Participants connected the code to visual forms, rhythm, poetry, media, culture, institutional values, and imagined audiences, often moving away from immediate questions of function. Surprisingly, the conventionally structured Website Evidence Collector generated far more associative remarks than the explicitly aesthetic Adder code: 73.6% of all annotated associations came from the second code text. Group 1 produced more than twice as many associative comments as Group 2, and association-building was distributed across all participants in Group 1, whereas in Group 2 it came primarily from Gloria and Jacques. Alan's trajectory was especially notable: he produced only one associative comment in the first session but five in the final session, suggesting that associative interpretation became more accessible to him over time.

Speculation about the author was another major form of non-teleological engagement. Participants treated code as the residue of decisions, values, humour, priorities, and imagined relations to an audience. Group 1 tended to imagine the author through play, virtuosity, aesthetics, and performance, while Group 2 more often politicised authorship, discussing surveillance, institutional voice, technological culture, and the values embedded in naming decisions. These conversations were necessarily unverifiable, but participants were often willing to remain with them. Even technically oriented participants later described such discussions as memorable or illuminating. At the same time, this willingness was uneven: speculative conversations were more frequently redirected toward technical explanation in Group 2, especially by Claude. Of the 17 annotated moments in which speculative discussion was minimised through technical explanation in Group 2, none occurred in session 2, when Claude was absent.

The method therefore succeeded in sustaining non-functional interpretation, but less clearly in sustaining multiple competing interpretations at once. Participants were generally comfortable entering uncertain discussions

about intention, culture, audience, and institutional meaning, yet these conversations usually moved toward some form of shared interpretation rather than preserving several possibilities indefinitely. Collective close reading did not eliminate the desire for closure; instead, it delayed closure long enough for associative, political, authorial, and poetic meanings to develop. Its effectiveness was therefore strongly conditioned by group dynamics, disciplinary habits, and the willingness of participants to let speculative claims remain unresolved.

5.4 Poetic Teleology

One of the more significant findings of the analysis was that the poetic interpretation did not only emerge when participants suspended function. Both groups repeatedly discussed poetic dimensions that were made legible through the technical qualities of the code itself. This was not a marginal behaviour, nor was it confined to participants with literary or media backgrounds. All participants engaged in this from of reading, and did so with surprising consistency. The second code text generated slightly more of these discussions, with 69.70% of recorded comments on the marriage of a functional and poetic quality being made during analysis of code text 2, leaving only 30.30% for code text 1. Within-group analysis shows Group 1 was even more biased, with 81.25% of these comments pertaining to code text 2, whereas Group 2 was more equitable, having 58.82% of their functional-poetic observations arising from code text 2. While both groups favoured code text 2 for this category, both code texts enabled participants to consider how technical aspects of code could also become poetic, aesthetic, affective, or interpretively meaningful.

This finding complicates the simple division between functional and poetic interpretation. Some features of the code texts appeared almost entirely non-teleological. The layout of *Adder*, for example, has no technical necessity at the level of execution. The compiler flattens the arrangement, meaning that the code's visual form in the editor serves no functional purpose in the running of the program. Its layout can therefore be understood as almost purely aesthetic: a poetic arrangement made for the human reader rather than the machine. By contrast, the layout of the *Inspector* class is shaped by the structural demands of TypeScript and object-oriented programming. Imports, class declarations, constructors, methods, and variable declarations appear in an order that is not arbitrary, but technically conventional and functionally meaningful. Yet participants still found poetic dimensions within this structure.

> **Gloria:** I think before that I was really trying to kind of view this 'artistic nature' of [the code], I still believe that there is an artistic nature, but [these sessions] helped me reimagine how an artistic or aesthetic nature is really fully perceived. [I was no longer looking for what] *my* preconception of what aesthetics is. [...] Before that I was trying trying to look at it broadly and see 'what's the shape?' [...] I was trying to see one format of poetry

> and then another format of code, and I think now I can see it a lot more as something that is maybe just different subjects of close reading. [...] So kind of like the creative endeavour of finding more efficiency.

Between the "one format of poetry" and the other "format of code", as Gloria describes, is a spectrum. Punctuating this spectrum are features such as variable and method names, which of course support functional legibility but *also* carry rich metaphorical, affective, or rhetorical force. A name such as *cookieRecorder* or a term such as *enrich* is not merely decorative, but neither is it exhausted by its technical role. While a rose by any other name may smell as sweet, any other name than 'rose' may produce a field of different associations. Evidently, this seems to be the same for code. Names may not alter an object's function, but they alter the relations through which that object becomes readable.

The distinction, then, is not between technical features and poetic features, but between different ways technical form can become available to poetic attention. In some moments, participants identified poetic qualities that appeared to exceed function, in others, they found the poetic force in the functional organisation of the code. We might describe these as teleo-poetic features: features whose poetic force appears through, rather than despite, their functional role. We see certain meta-categories appearing of teleo-poetic features, which we have attempted to formalise. Structure and layout, strategy, intentionality and runability, naming and word choice, and finally, infrastructure and context. These were the sites where execution and expression were difficult to separate. Of the roughly 33 explicit comments describing how a functional component was poetic, we saw these categories referred to 11, 10, 7, 6, and 6 times, respectively. Astute readers may notice this does not add up to 33, this is because some instances fit into more than one category. For example:

> **Charles:** Yeah, I think that set of variables in the constructor is very indicative of what you, *[Alan]*, were saying about this being very specific, very pointed, in what it's doing. Like, normally it would just be like: constructor and recorders, you know, like just all the recorders, everything. Like bad coding or lazy coding, really. [...] but this [code] is very like: we only need *these* things and so we're only pulling in these things. It's clean. It adds more lines, but that extra rigour is needed in order for it to be, you know, something that would be used by the EU, like at that level.

This quote from Charles on the decision to be precise with constructor variables is referring to both strategy (which encompasses things such as efficiency and optimisation), but also structure and layout (which would involve the necessity of a constructor structure itself). Participants discussed how the structure of the code shaped the way it could be read, how naming decisions altered tone and readability, how repeated declarations produced

a rhythm, how line breaks interrupted cadence, and how technical hierarchy could give rise to narrative or affective associations. The poetic and the technical were often not two separate layers, but the same feature read under different modes of attention.

This also complicates participants' own accounts of technical conditioning. Several technically trained participants suggested that their familiarity with programming made it harder for them to see alternative or poetic meanings in the code. Yet the sessions show that these same technical frames often became the means through which poetic interpretation became possible. Alan, for example, repeatedly used his knowledge of syntax, semantics, function, and implementation to produce or extend poetic readings. Rather than preventing poetic attention, his technical understanding sometimes gave him a tangible framework through which poetic qualities could register. The solution, approach, or structure of the code could become emotionally or aesthetically significant precisely because he understood what was technically being achieved. In response to having the code read, and re-read with different rhythms and cadences, Alan offered this insight:

> **Alan:** [Facilitator], I think your research is onto something because going line by line, I'm starting to really understand what's going on. Yeah, because it's first trying to...it first inspects cookies. After it does that, based on the cookies it gets the host and then it inspects them one by one. So I do understand everything up until the moment that we are here.

These comments were also a referral to a conversation had before the session began, when Alan had proposed sharing poetry with one another. He discussed how poetry was often a memory device for long epics and oral stories. Here, he relates the poetic reading of code to the poetic metre of epics as a means to improve *functional* understanding.

This is something Michael engages with a lot as well. As mentioned before he often combines his technical offerings with a poetic openness.

> **Michael:** [A] lot of good coding practice is kind of in making it more readable for other people, because there are so many ways to implement any block of code, but a lot of the ways are also more readable by people, you know, [like] properly indenting things so that you can clearly kind of chunk it and arrange it well. That is a consideration that is for human readability. The question then: is it a purely aesthetic decision or is it a functional one? Especially if you're designing in such a way that you want a person to be able to read and understand the code. It does serve a functional purpose, I would say.

Often in dialogue with his colleagues, Michael acknowledges that software conventions themselves may not be antithetical to poetic readings but may be poetic themselves. This is an example where one sees the technical participants audibly and explicitly reorienting their perspectives to be less *absolute* in their technical interpretations. Often, the technical participants

are able to find the poetic in the technical frameworks present. Since their background is technical, they use this as such a frame. It becomes their barometer for poetic charge.

For the humanities and media backgrounded participants we also saw a phase shift in their poetic registers. Gloria makes a number of explicit remarks about her conceptualisations of what is poetic changing form. One was shared at the beginning of this section, but to share another, perhaps more evocative one;

> **Gloria:** I also think it's interesting though because again I feel like we're faced with this kind of clear idea of *aesthetic* and *function* just not really being as separate as we're perceiving it because repetition, for example, I mean speaking from a poetics perspective, repetition is something that I actually view as again something quite functional within poetics in that it is used in order to emphasize something as well. Not, of course, not always, but it can be used very much to emphasize something, to kind of like understand, yeah, understand the importance of something. So I think therefore...
>
> [...]
>
> **Michael:** Like the aesthetic is almost giving the function or aiding in the function.
>
> **Gloria:** You could say its almost like a tool for the function.

Laura, in Group 1, shares her perspective;

> **Laura:** ...coming back to the code, it's the question of the relation of the functionality and the aesthetic. That is, for me, kind of hard to wrap my my head around on [...] because there can be, I think, something poetic in that like it's very elegant from a functional perspective. That can be considered something poetic; to come up with something new and innovative in a form. Then also the relation between aesthetics and function, I can imagine can be poetic and then you can, if you [want], only look at the aesthetics part, not looking at the whole thing but picking it apart randomly wherever you like and just focus on the word 'origin' or 'infinity'. Even that, that word itself can be it's an open container where you can fill it up with relation to other words and make make your own poetry out of this code even if it was not intended by the author.

The significance of this finding is that collective close reading did not require participants to abandon technical understanding in order to read code poetically. Instead, the most productive moments often occurred when technical and poetic interpretation were held together, when one becomes a "tool" (as Gloria put it) for the other. The opposition between teleological and poetic reading cannot be maintained too rigidly. The poetic does not always stand outside the technical; it inheres in the way technical forms are

named, organised, repeated, constrained, and made available to human and non-human readers.

5.4.1 Summary

One of the clearest findings was that poetic interpretation did not emerge only when participants moved away from function. In both groups, technical features themselves repeatedly became poetic material. This happened more often with the Website Evidence Collector than with Adder: 69.7% of the explicit functional-poetic comments concerned the second code text, rising to 81.25% in Group 1. Participants repeatedly found poetic force in structure and layout, strategy, naming, runability, and infrastructural context. Features such as cookieRecorder, constructor structure, repetition, indentation, and technical hierarchy remained functionally meaningful while also becoming aesthetic, affective, rhythmic, or metaphorical. The distinction was therefore not simply between technical and poetic features, but between different ways the same technical form could be attended to.

This shift occurred across disciplinary backgrounds. Technically trained participants such as Alan and Michael often used their knowledge of syntax, implementation, readability, and software convention as the very basis for poetic interpretation; Alan even found that rhythmic rereading helped him understand the program's function more clearly. Conversely, humanities and media participants such as Gloria and Laura began to reconsider their own assumptions about aesthetics, recognising that efficiency, repetition, elegance, and functional organisation could themselves be poetic. Collective close reading therefore did not require participants to abandon technical knowledge. Instead, some of the most productive moments emerged when technical and poetic interpretation reinforced one another, making function itself available as a site of poetic attention.

5.5 Poetic Categories and Features

This section discusses the most common reoccurring poetic features that precipitated from the collective close reading sessions. When combing through the transcripts, we annotated instances of poetic interpretation and attempted to categorise them. The resulting section is composed of the categories which collected enough examples across different participants, sessions, and groups to be identified as an ossified poetic feature of the texts being read.

We observed four main categories. The first discussed is sensory and affective encounters, which considers emotional, visual and sonic features of the code. Next we discuss textual and linguistic affordances, which deals with the employment of text, such as its wording and tone. Following this we look at structural and compositional form, which refers to the layout and relationality of the internal structure and its effects. And finally we look at contextual and political reflections, which will analyse how the environment and positionality of the code texts influences its meaning and analytical depth. Each of these sub-sections will have a summary to allow the reader

to choose the level of analytical depth with which they wish to engage.

5.5.1 Sensory and Affective Encounters

Affective Responses

Affective responses describe participants' emotional or phenomenological reactions to particular aspects of the code. As such, affect was rarely a stand-alone poetic feature, but more often appeared in response to, or in parallel with, one of the other poetics dimensions. The most common pairings were visual and textual qualities, audibility, rhythm and cadence, and broader associations emerging from commentary, comparisons to poetry, and naming decisions. Affect, therefore, often provided an immediate indication that participants were registering something beyond the exclusively functional operations of the code.

There was a slight increase in affective responses over the three session in both groups. Interestingly, in both groups the participant with the most affective comments was a technical participant (Charles and Michael), although these participants were followed by almost all of the non-technical participants in their respective groups. In Group 1, the other technical participants also contributed reflections on affective capacity of the code, however, in Group 2, the other technical participant, Claude, did not appear once in the collection of annotated affective comments. This is notable because the affective comments were not restricted to poetic or aesthetic qualities; simply appreciating or criticising the code through an emotional register was sufficient to constitute an 'affective response' within our criteria. Claude therefore did not merely refrain from describing the code as beautiful or poetic, but did not appear to register the code through *any* explicitly affective register at all.

> **Gloria:** I thought it was quite human that they added 'this' [referring to the object reference keyword *'this.'*]. So *"const cookies ="* calls *"this.inspectCookies"*, it was kind of like the code was utilizing some kind of...maybe like an awareness of the system. Awareness of like a context. So being able to like look at a word "this" and think: okay, I connect that to something else, which I think is like a very, in my head, [...] relational understanding, [it] is a very human thing to do, and I kind of assume that in code every single word is very significant...

In Group 1, affective responses initially emerged as immediate reactions to the code, such as "this is neat" or "this is gross";

> **Charles:** But, like, me looking at it, [I think] oh this is gross.
>
> ...
>
> **Charles:** I'm seeing the elegance Okay, now I'm seeing the elegance to it. Like it takes working through it.

As the group developed, these responses increasingly became attached to more tangible features of the code, such as particular poetic instance or association. Group 2 exhibited a similar movement, although affective responses were more commonly connected to an identifiable feature from the beginning. This suggests that affective engagement was often an initial mode of encountering the code, which could subsequently become articulated through more specific poetic or associative interpretations.

Visual Aesthetics

Visual aesthetics were most commonly discussed alongside inspired associations, particularly when participants described what a particular arrangement reminded them of visually, and alongside affective responses to the visual appearance of the code. Visual qualities also inspired considerable discussion of authorial intentionality, which in turn led to broader discussions of software conventions. There were often attempts to explain away visual pareidolia through technical explanations of structural requirements. However, these were frequently followed by, or even overshadowed by, a recognition that formal necessity and visual expressivity could coexist. Visual aesthetics therefore provided another instance of the previously discussed union of poetic and technical aspects. One example was a quote shared previously from Claude:

> **Michael:** [on the imagery of Adder] It feels a little bit like misdirection and I think that misdirection could be part of the commentary.
>
> **Facilitator:** Claude, you made a face earlier.
>
> **Claude:** Yeah, I'm not very convinced that the discrepancy between this being a simple adder and the programme being a little bit more complex, I don't think that the difference between them is that deep to be honest. An adder is an adder, whether it's a programme or a logic gate, [...]. I don't think the programmers intended that gap to be there. Or think about the gap.

For both groups, the majority of the commentary on visual qualities concerned the first code text. However, visual aesthetics also provided an interesting insight into the idiosyncratic nature of poetic engagement. For example, Ada and Jacques, did not make any annotated comments concerning the visual structure or layout of the code. This was notable because both were otherwise highly engages in poetic analysis, and both contributed extensively to other dimensions of code-poetic interpretation. Their absence, therefore, does not seem to indicate a general unwillingness to engage poetically, but rather suggests that particular poetic affordances are encountered differently by different readers. What makes code poetic, or at least what makes a particular poetic quality meaningful, appears at least partly idiosyncratic.

This was particularly evident in Group 2, where even Claude made multiple comments concerning the visual qualities and affordances of the code, despite his relative reluctance to engage with many other non-technical interpretations.

> **Claude:** Oh, there is one thing I can comment on. So if I were to programme such a thing [as the Adder code, particularly the cipher] in the define method, I would start very structurally: so let's make the first letter a capital letter and then the rest of them three lowercase letters, and then go through it very structurally so you don't get the same words accidentally twice. But in this case they did it all over the place. They did...there was no structure to encode the different symbols and if I would do it I would do: an uppercase letter would be a one and a lowercase letter would be a zero and in that way you can make sure you don't copy a symbol you already did.
>
> **Michael:** I do feel it would be odd if there were no structure though.
>
> ...
>
> **Michael:** But I mean back to the arrangement of like the order [of the cipher]. It feels weird that there would be no order, but I am not able to find kind of an object. Order. [Groups spends some time reading] No, it doesn't.
>
> **Claude:** Because as a human reading it you probably won't. Well you'll need to make notes of it.
>
> **Michael:** So what I'm thinking is I think they did maybe originally write them in an order and then swap it around so that they could have this kind of cascading thing on the side.
>
> **Claude:** Oh right, yeah that's also impossible. Right, yeah.
>
> **Michael:** Because one thing that we haven't really talked about is, I guess, what it looks like to have this shape on the side for the defines, right? They all go from longest all the way down to shortest. So those are ultimately ordered in some way. But um but the mains themselves don't appear to be.
>
> **Claude:** I don't know, it could be just a stylistic choice. Um just to have it random.

This exchange between Michael and Claude provokes an important point worth emphasis; we do not intend to present or imply Claude is a participant who resists poetic interpretation on principle. What we want to highlight is that he is capable of identifying particular poetic qualities when they emerge through a dimension to which he was already comfortable attending. In this exchange, Michael and Claude discuss the reason for the visual arrangement of the substitution cipher's key. Claude mentions this is something he feels comfortable addressing. Clearly this element sits at an intersection of aesthetics and logics that is within a realm he feels confident engaging with.

Visual qualities continued to appear in comments from technical participants, although these were often brief, passing observations. When pushed further to explore these responses, they tended not to elaborate extensively. It is difficult to determine from the data whether this indicated a lack of perceived significance or simply a reluctance to pursue an interpretation that sits outside of their established technical frame.

Audibility and Reading Affordances

Audibility and reading affordances concern how the participants interpreted and pronounced the symbols of the code. The symbols commonplace in a piece of code text are unfamiliar in both quotidian speech and conventional poetry, while, simultaneously, possessing highly specific meanings within a programming context. In this context, familiar punctuation can therefore metamorphose into something else entirely. For example, sometimes, symbols swap their traditional meanings; in coding languages, a semi-colon often serves as the marker for the termination of a statement, the role typical to a period/full-stop in literature, while the period can operate as a conceptual connector, indicating the relationship or connection between two elements, something usually done by a semi-colon. These transformations provide a diversity in ways of how code might be uttered, pronounced, and sonified.

One immediately capturing detail, is upon sharing the Website Evidence Collector code text, during a silence wherein the facilitator encouraged to simply get acquainted with the code, Charles' first response was to read out loud, rhythmically, some lines from the code:

> **Charles:** A lot of recording going on. First party *checker*, [emphasis on checker, pause] cookie **recorder**, [emphasis on recorder, pause] beacon *recorder*, [emphasis on recorder, pause], host *recorder* [emphasis on recorder, pause].

What is interesting is that this was Charles' immediate reaction; to read the code. But furthermore, he instinctively added a level of rhythm and order to the syllables of the lines without any direction.

Peculiarly, every single participant read the text differently. Claude made many explicit references to the breadth of possibilities in articulation, however his comments were in the form of doubt, second-guessing, or questioning. Jacques, by contrast, had the most to say in this category. He also frequently second-guessed his own recital, but did so more often in dialogue with the interpretations of others. His engagement appeared to come from an enthusiasm for exploring different possibilities rather than from a desire to establish the correct pronunciation or reading.

> **Jacques:** I would read this comment differently because, [like I said], it makes more sense reading it like this trying to split all the words from each other because when we perceive a text we want to connect all those words that don't flow, that's how we read the code last week. But I would also, today, read it

> like [Claude] did, maybe. And, I would have much more emotions on the comments because that's specifically for the human. But I guess it's still good to put emphasis on the disconnected words. [It] also says something; they exist on their own but also within relation to each other, but not in a way that we want to perceive a text not with a perfect flow.[But] just because of their functions, two words can come together unlike a natural conversation.

Jacques often went into monologues like this, reflecting on the purpose and affects of different styles of reading. In both groups, as with visual qualities, particular participants stood out as especially interested in the acoustic dimensions of the code. Laura and Jacques were almost the only participants, apart from the facilitator, to invest sustained effort into exploring the sonification decisions of others. Their contributions were less concerned with establishing a correct way of reading the code aloud and more concerned with exploring what different modes of reading might make possible.

Audibility and reading affordance were one of the clearest dimensions in which different interpretations were able to coexist. Participants seemed more comfortable accepting another person's reading of the code than they were accepting or challenging their broader interpretations of what the code meant. We see this in Jacques' above quote. Almost everyone across both groups developed different ideas about how particular symbols or sequences should be articulated, and these decisions appeared to be remarkably uninfluenced by one another. There was almost no consistency in how the code was sonified across participants. Rather than converging on a shared pronunciation, participants seemed content to retain their own ways of reading the symbols.

We can reflect on this by reviewing the different styles from the participants in Group 1. Alan was first to read the code aloud. When reading the opening to the Inspector class (the group decided to skip the imports), he read each line with a similar cadence and inflection. It was slightly musical, with an upward inflection at the end of each checker/recorder word ending. Most uniquely was his reading of the colons, choosing to read it as the words 'of type'. So, *private collectorOutput: CollectorOutput* becomes "private collector-output of type collector-output". He paused at each new line. He didn't change his volume or tone compared to his regular, conversational speech. Laura chose to re-read the same section. After a short hesitation about how to read certain aspects like tabs, spaces, and new lines, she began in much more theatrical, soft, intentional voice. She read everything literally, she pronounced 'new line' upon a new line, 'comma' upon seeing a comma, 'curly bracket' upon seeing a brace ('}'), and so on for every symbol and white space form.

Ada read the next passage. She commanded attention with how she read the function name, 'run'. She took a long pause and let the word hang in the air. She spoke it like a direction or an order. Ada also read everything literally, articulating symbols by their names. She omitted tabs

but did remark on new lines. However, her most signature quality was a very strong sense of rhythm and pacing. She made intentional and noticeable pauses at each white space, between the words 'const' and 'cookies' for example. But at other times she would musically flow through terms, such as 'this.inspectCookies();' which became a melodic 'this-dot-inspectCookies [slight pause] bracket-open [slight pause] bracket-closed [slight pause] semi-colon [long pause]'. In the field notes from the facilitator it remarks that she nodded her head along with the self-organised rhythm.

Charles read the next section, unprompted, choosing to do so when the conversation arrived at this section and its function. He imposed a very slam-poetry-esque style to the content. He had inflections and emphasis in moments that are perhaps not immediately sensible, for example, running through a line but inflecting and paused at equals, and speeding off again. He chose to not articulate symbols, leaving out almost all non-lexical characters aside from periods which he read as 'dot'. He also was inconsistent with his reading of the equals signs, sometimes emphasising them and other times omitting them entirely.

Finally, Marvin, the last to read, tried a couple ways to enunciate the symbols, restarting twice. He read it first with extreme literal articulation, then once only saying the double 'equals' symbols ('==') and also the double 'and' symbols ('&&'), then finally with none of them. He also narrated much of this thought process and explained his decisions much more than everyone else. He used a far more conversational tone than Laura or Ada, but was much more rhythmic than Alan, even remarking:

> **Marvin:** I feel like I'm beat-boxing with symbols.

Taken together, these readings suggest that audibility provided one of the clearest sites in which code could resist interpretive closure. The symbols themselves remained technically determinate, yet their vocalisation was not; punctuation, spacing, repetition, and syntax became materials for rhythm, emphasis, hesitation, and performance. Reading the code aloud therefore did more than reveal an acoustic dimension already present in the text. It made visible how differently the same technical inscription could be inhabited by different readers. In this sense, audibility became a particularly direct example of poetic attention at work: the code did not cease to function as code, but its execution was joined by another form of realisation, one produced through voice, cadence, and embodied interpretation.

Rhythm and Cadence

Rhythm and cadence are closely related to audibility, although they represent only one manifestation of the latter. The relatively high frequency of comments concerning audibility and reading affordances, compared with the smaller number explicitly coded as rhythm and cadence, suggests that participants encountered acoustics qualities in the code that extended beyond rhythm itself. This was particularly apparent in Group 2, and especially in Jacques' engagement with different possibilities for reading the code aloud.

Some rhythmic observations, however, were not explicitly articulated by participants. Rather, they emerged through our observations of the way participants actually read the code aloud, particularity the stylised rhythms and cadences they adopted while reciting. These observations are therefore somewhat different from the other poetic features discussed in this section; rather than participants explicitly identifying rhythm as a quality of the code, the rhythm sometimes became apparent through their performance of the text, which can be seen from the analysis of Group 1's reading techniques in the previous section.

Rhythmic references appeared almost exclusively in the second code text. They were also the source of a number of epistemic adjustments, as participants began to recognise rhythm and cadence within the code that they had initially approached primarily through its functional structure.

> **Michael:** Yeah, because that is another repetition that I hadn't really thought about. Because, like when I was talking about repetition earlier, I was really kind of meaning the private, private, private, private, private, private. But no, very, very much so, the naming the each variable the same as the actual input class is definitely a choice that might make it more also human readable.
>
> **Jacques:** I feel like it's not done for the aesthetic reason but almost fully for the functional reason, so that's why I'm seeing it as not like an artistic choice of the repetition, sure, but like a functional one.
>
> **Michael:** Yeah, yeah, yeah. But the thing is I don't agree that...I don't think it's an *artist* choice, but I do think it is an *aesthetic* choice, that the aesthetic choice is *aesthetic function*.
>
> **Jacques:** Functionally aesthetic.
>
> **Michael:** Exactly. Exactly, because the aesthetic serves a function. I don't think it was necessarily written with the intention of being viewed as artistic, but I do think the aesthetic of, you know, the repetition making these two the same was very intentional.

Across both groups, production of this category was relatively evenly distributed among participants, with the exception of Claude, who did does not have any contributions to this category.

Summary

Across both groups, affective and visual responses provided some of the earliest ways participants moved beyond purely functional readings of the code. Affective comments increased slightly over the three sessions, and, notably, the participants producing the most affective remarks in each group were technically trained: Charles in Group 1 and Michael in Group 2. Claude, by contrast, produced no explicitly affective comments at all. Visual interpretation centred mostly on Adder, whose unusual layout made

aesthetic form difficult to ignore, although participants responded very differently to it. Ada and Jacques, despite being highly active in other forms of poetic interpretation, made no annotated comments on visual structure, while even Claude engaged with the visual organisation of the code when it intersected with questions of logic, ordering, and implementation. Visual aesthetics therefore emerged as a strongly reader-dependent affordance rather than a universally salient poetic feature.

Audibility produced an even clearer plurality of interpretation. Every participant developed a different way of reading the code aloud, varying in whether punctuation, whitespace, brackets, or operators were spoken, and in the rhythm, emphasis, tone, and pacing imposed on the text. Participants appeared comfortable allowing multiple sonifications of the same inscription to coexist without establishing a correct pronunciation, making audibility one of the clearest examples of sustained interpretive plurality in the study.

Rhythm and cadence emerged closely alongside these practices, particularly in the Website Evidence Collector code, where repetition in declarations, variable names, and structural patterns became increasingly audible through recital. Michael and Jacques came to describe repeated naming structures as "functionally aesthetic". Repetition could improve human readability and serve a technical purpose while simultaneously producing an aesthetic or rhythmic quality. Across these categories, sensory and affective engagement therefore did not consistently arise by leaving technical structure behind; often, visual form, rhythm, audibility, and affect became perceptible precisely through sustained attention to the forms produced by technical organisation.

5.5.2 Textual and Linguistic Affordances

Textual Qualities

Textual qualities refer to features such as capitalisation, grammar, pronouns, spacing, indentation, tabs, and other textual conventions through which code is presented. This category is closely related to linguistic affordances since the language in which code is written determines what kinds of textual manipulations are possible. However, these comments were generally concerned with the textual quality of the code itself rather than the specific affordances of the programming language.

Both groups produced the most textual reflections during discussion of the second text, 70% for Group 1 and 58.33% for Group 2. Textual qualities were among the categories most frequently used to bridge toward traditional poetry. Participants would often begin with an observation about capitalisation, spacing, grammar, repetition, or another textual feature before connecting it to a poetic convention or form.

> **Alan:** I mean poetry [is] usually also—in comparison to prose text—is like in blocks with paragraphs. So, especially this first side [of Website Evidence Collector] could be like [the methods] are verses, because they are very distinctly apart from each

other.

Facilitator: Like stanzas for a poem almost?

Alan: Yeah! Yeah.

In this sense, textual qualities provided a relatively direct point of contact between the formal properties of code and participants' existing understandings of literary text. The category was discussed relatively evenly across the two groups.

Naming Decisions

Naming decisions constituted a more contentious form of textual affordance, particularly in Group 2. Alternative associations with variable names were frequently offered but were often met with attempts to redirect the discussion toward programming conventions rather than being explored as interpretative possibilities.

> **Gloria:** There's intentionality between why they have chosen these words and yeah, they might not choose a word like monitor, maybe.
>
> [...]
>
> Or yeah, surveillance or something. It's 'collected'.
>
> [...]
>
> **Michael:** I think it's also important to consider other aspects of the context, in that I do think a lot of these words, inspector, recorder, output, checker, they have connotations and meaning within programming spaces. And I think there are conventions for, you know, what you should call specific things based off of their function and then this is, I think, adhering to those conventions.

Consequently naming associations in this group were less likely to develop into sustained speculative discussions and more likely to collapse back into explanations of computer science history or convention. However, interestingly, Group 2 still produced far more instances of comments regarding names (17 versus Group 1's 4), our point is simply that these points came and went, rarely leading to in-depth analysis or reflection.

The locus of consideration for Group 1 differed quite a lot from Group 2. Although variable names were discussed (exclusively in the first code text) these discussions were less frequently concerned with *why* a programmer might have selected a particular name. Instead, the participants more often considered the affective or associative qualities produced by the name itself; how the repetition of a word felt, what it reminded them of, or what associations it generated. The difference was not only one of distribution and frequency but one of interpretive orientation. Group 1 most often treated names as textual and affective material, whereas Group 2 more frequently treated them as evidence of authorial or disciplinary perspective.

The distinction is reflected in the distribution of the category, which appeared approximately four times as often in Group 2 as in Group 1. The greater frequency in Group 2 therefore does necessarily indicate a greater willingness to engage poetically with naming. Instead it reflected the extend to which naming became a site of discussion about technical convention, intentionality, and institutional language.

It is also important to distinguish variable names form other technical terms such as "cookie" or "localStorage". These terms were not necessarily selected by the programmer who's work is being examined and therefore not coded as naming decisions, even though they generated extensive discussion. Such instances are instead addressed in the categories of broader commentary, programming conventions, and authorial speculation.

Linguistic Affordances

Linguistic affordances provide perhaps the clearest indication of the extent to which Group 2's discussion centred on the relationship between functionality and poetic possibility. Participants in Group 2 frequently considered how the particular programming language shaped what could be expressed, associated, or interpreted within the code. These discussions often emerged alongside observations about textual qualities; participants would identify a textual or poetic feature, consider what it allowed them to associated or interpret, and then connect this possibility to the affordances of the programming language itself, particularly TypeScript or C.

> **Michael:** I think another interesting thing to consider for C and I guess what it affords is...to my understanding, if you really wanted to in C, and I guess C++ as well, you can just write all of your lines of code in one line and just make sure all your quote-unquote 'lines' are separated with a semicolon, as they would if you indented, which also means that at the end of every physical line you don't have to put the semicolon because if you look throughout we can see *'MAIN'* with just uppercase *M* everything else lowercase is the semicolon. And if you look through, you know, you can find a few of those, and they are in middle of the physical lines. And that is again another kind of affordance of the programming language.

Group 1 made no annotated comments concerning linguistic affordances in relation to the first code text. Group 1 generally approached the first text through its aesthetic properties, relatively removed from the structural and linguistic affordance of C itself (with some exceptions such as their discussion of white-space). Group 2, by contrast, more consistently considered how the technical structure of the language itself contributed to what could be made poetic. Overall, Group 2 produced twice as many comments in this category as Group 1 (12 versus 6, respectively).

Unlike some of the more isolated observations in other categories, linguistic affordances frequently became extended points of discussion. This was relatively evenly distributed across the participants in Group 2, with

all members engaging with the relationship between programming language and poetic potential. In Group 1, the discussion came primarily from Laura and Ada, with some contributions from Alan. The participation of both technical and non-technical participants in this category is significant in that it demonstrate that linguistic affordances were not exclusively accessible through technical expertise. Rather, participants could approach the programming language as another formal material through which poetic possibility emerged. This also indicated that linguistic affordance is an avenue for the dissolution of disciplinary boundaries and an expansion of the interpretative authorial privileges within code.

Summary

Textual and linguistic affordances became one of the clearest points at which participants connected code to more familiar literary forms. Across both groups, textual observations were more frequent in the second code text, with participants discussing capitalisation, spacing, grammar, repetition, indentation, and layout before relating these features to concepts such as verses, stanzas, or poetic structure. Naming decisions generated a more contested form of discussion. Group 2 discussed names far more often than Group 1, but these conversations frequently returned to programming convention, intentionality, or disciplinary language rather than remaining open as poetic interpretation. Group 1, by contrast, more often treated names as affective or associative material in themselves. The difference was therefore not simply quantitative, but concerned what participants believed names were evidence of: textual feeling and association in Group 1, or authorial and technical positioning in Group 2.

Linguistic affordances extended this further by drawing attention to the programming language itself as a condition of poetic possibility. Group 2 produced roughly twice as many comments in this category as Group 1 and more consistently discussed how the structures of C and TypeScript shaped what could be expressed, arranged, or interpreted. These discussions often moved from a concrete textual feature to the formal rules that made it possible. For example, C's treatment of semicolons, line breaks, and whitespace were repeatedly brought up. Importantly, this was not only a technical-participant concern. Both technical and non-technical participants engaged with the programming language as a formal material. Linguistic affordance therefore became one of the clearest sites where disciplinary boundaries softened, since understanding the technical structure of the language could itself become a route into poetic interpretation rather than a barrier to it.

5.5.3 Structural and Compositional Form

Structural Composition and Layout

Structural composition and layout was the most commonly appearing site of the union between poetic and technical dimensions. Unlike visual aesthetics, where the visual arrangement of code could sometimes be con-

sidered independently of its function, these discussions concerned the ways in which the technical organisation of the code itself could produce poetic effects. The was also one of the areas in which technical expertise was most frequently offered in order to deepen or develop a poetic interpretation. Rather than technical explanation displacing poetic reading, participants sometimes used an understanding of the code's structure to make further interpretive possibilities available.

One recurring example was *worlding*. Both groups compared the structural organisation of the second code text to forms of narrative or world-building, with Charles and Jacques in particular discussing the way that the arrangement of function, variables, and constructors could be read as establishing a kind of universe of internal world logic within the code;

> **Jacques:** The reason that we are able to understand the code is because, coming back to the metaphor, this whole page is setting the world of what's going to be happening in the code and when you understand how these um like classes are being set up, you see what is gonna be happening in the code. And unlike putting all these classes in the later sections, setting them up properly in the first section works in a way to tell beforehand what's gonna come basically. So introducing the characters in the landscape.

This category appeared entirely in the second code text. This is significant given that the first code text was also discussed extensively in terms of its visual arrangement, but its layout was not generally interpreted through its technical or linguistic embeddedness. Instead, its arrangement was treated primarily as an aesthetic feature. In the second text, by contrast, participants' interpretation of structure were inseparable from an understanding of what that structure was doing teleologically. The poetic quality therefore emerged not despite the technical structure, but through it. The organisation required by the programming language and object-oriented conventions became part of what participants were able to read poetically.

This was also one of the clearest examples of technical knowledge functioning as an interpretive resource. Participants with technical expertise could explain why a particular structure appeared where it did, but this explanation did not necessarily settle its meaning. Instead, technical explanation could become another layer through which the structure was interpreted; a function could become a stanza, a sequence of methods could become a progression, or the organisation of the class could become a narrative architecture.

Repetition

Repetition was also considered exclusively in relation to the second code text. This is somewhat surprising given that the first code text consists, at least visually, of extensive repetition of the same word. Participants did not, however, dwell substantially on the poetic significance of this repetition. While there were occasional remarks concerning the experience of becoming

lost within the repeated code, repetition was not generally isolated as a poetic feature in its own right.

Instead, repetition became meaningful when it appeared within the structural conventions of programming. The repeated variable declarations within the constructor, for example, were understood simultaneously as a consequence of the organisation of the programme and as a formal repetition with poetic significance.

> **Charles:** SLAM code. That's what I do when I submit it. But yeah, no, it's like, "let's define the cookies and local storage and beacons and then use those to figure out the hosts, blah blah blah, and then use those hosts to define the hosts object of the constant hosts, this dot inspect hosts. And then from there. Oops. It's this collector output web socket log equals this web socket recorder collected web sockets." That's the slam poetry.
>
> **Alan:** And also like this [pointing], where we had in the beginning the repetition of certain words? That does create an inner logic of relation; like a recurring object that gets loaded with new meaning each time it appears the next time.

The repetition was therefore not simply perceived as repetition; its placement within a technical structure contributed to its interpretability. As with structural composition more broadly, this suggests that the poetic quality participants identified was produced through the intersection of technical form and poetic reading.

Furthermore, there was a surprising absence of discussions on repetition in relation to audibility or rhythm. Instead, its significance appeared primarily visual and structural. This is particularly notable because repetition was repeatedly identified in the exit interviews as one of the most memorable poetic qualities of the code.

> **Michael:** I think immediately, off the top of my head, repetition. And, just, different kinds of repetition. Right? Because, of course, every time you repeat a variable, because you need to use it again, that is repetition. But also, structural, the way that your...like, lets say we're just declaring a bunch of variables, you also have multiple lines in a chunk that are following the same structure, one after another. For example, with the TypeScript example, when they were declaring the variable, and the 'of type *whatever*', and its basically the same. The variable was then camel case, and the actual type was capital-capital. So, things like that. Repetition was, kinda, the big [poetic feature] that stood out to me.

Despite there only being 13 coded instances of comments explicitly referring to repetition, almost all participants mentioned similarly to Michael's above quote. Repetition was an important factor in relating code to traditional poetry. It was a memorable poetic feature that both emerged from and transcended its functional telos.

Narratisation

Narratisation refers to moments in which participants experienced the code as producing or implying a narrative through its organisation, sequence, or progression. In a broad sense, these moments also involve an affective component; participants were responding to the experience of encountering a narrative within a text that was not conventionally understood as a narrative. However, narratisation is distinguished here from the broader category of affective response or associations because participants were not simply reporting how the code made them feel or what is reminded them of; they were identifying narrative structured within its technical organisation.

Narratisation was almost always discussed in relation to the layout and structural composition of the code. Participants would move from noticing the organisation of functions, variables, and operations to describing these arrangements in narrative terms.

> **Ada:** I think all of it is like uh storytelling so we define the scenario situation And then we define the characters and then we move on with the story like what they are doing, how they are interacting. And then we see what happens.
>
> [...]
>
> These variables, functions [are the characters]. Like functions could be the actions, what they are doing, and then the variables would be these characters.

This was particularly significant in relation to epistemic adjustment. Technical participants, like Ada in the above quote, who had initially approached the code primarily through its functional register sometimes began to recognise that the same organisation could support another mode of reading. The code could still be understood as a sequence of instructions but it could simultaneously be read as a sequence of events, relations, and developments.

Summary

Structural composition was one of the strongest sites where technical organisation itself became available for poetic interpretation. This appeared entirely in discussion of the Website Evidence Collector, where participants treated constructors, methods, variables, and the ordering of the class not only as functional necessities but as compositional devices. Both groups compared this organisation to narrative or world-building, with participants describing the code as establishing characters, actions, relations, and an internal logic before those elements were put into motion. Technical expertise frequently deepened these readings: understanding why a structure existed did not close interpretation, but instead made it possible to read a function as a stanza, a sequence of methods as a progression, or the architecture of the class as a narrative world.

Repetition and narratisation developed from the same structural attention. Repetition became significant not simply because elements reappeared, but because repeated declarations, naming patterns, and formal

structures accumulated meaning through their position within the programme. Although only 13 comments were explicitly coded as repetition, almost all participants later identified it as a memorable poetic quality, and most participants described it as the feature that stood out most strongly to them. Narratisation similarly emerged when participants re-read technical sequence as narrative sequence. Variables could become characters, functions actions, and program flow a progression of events. In both cases, the poetic quality was not detached from function; it arose through sustained attention to the organisation that made the code work.

5.5.4 Contextual and Political Reflections

Separation of Audience

The separation of audiences was one of the ways participants moved from the immediate qualities of the code toward the questions of who the code was written for, who was expected to understand it, and what it means for code to have an audience. In Group 1, this discussion was initially dominated by technical participants. Early conversations focused on whether the code was more efficient or comprehensible to the computer, drawing on the authors' own comments about making a comfortable and understandable piece of code for the machine.

> "My goal was to create the fastest possible C program. To that end, I made three critical observations:
>
> 1. If there's one thing computers are good at, it's math.
> 2. Simple operations take less time than complicated ones.
> 3. Every C programme seems to contain the word "main".
>
> Based on #1, I knew that the Fastest programme had to be one that performed addition. From #2, I reasoned that it ought to directly manipulate the bits, rather than wasting time dealing with bloated, high-level, fuzzy-logic, artificial-intelligence, neural-net, client-server, object-oriented abstractions like the C language "+" operator. And it was obvious from #3 that the programme should resemble, as closely as possible, a long sequence of the familiar word "main" repeated over and over, so the computer would be comfortable running the programme and wouldn't get distracted dealing with unfamiliar variable names." (Downs and Makarios, 1995)

This quickly precipitated reflections on the ironic dimensions of the code text, as participants contrasted the apparent difficulty of the code for a human reader with its supposed readability for the computer, opening into broader commentary on computer science or software engineering culture and the positionality of the programmer.

The reflections on who is code intended for continued into the second code text; discussions on legibility, intended audience, and balancing efficiency of language and operation with readability were carried over to

this new context. Whereas the first code text mostly generated a duality between the computer and an engineer (or human in general) as the two potential audiences, the second was more inclined to compare engineers to non-engineers.

> **Laura:** But is it poetry for the computer then?
>
> **Charles:** No, it's poetry for the people that the computer can also run.
>
> **Laura:** But it's functional as well as form. But if the person who wrote it said, "okay, this is to soothe the computer in a way", then it might just be that the computer is being being considered as something that has a relation or some kind of a...that the computer has a different underlying layer of meaning towards [the code] that could make it feel uncomfortable with something that doesn't know. So it does attribute to the computer something.
>
> **Charles:** I think they were being cheeky.

In Group 2, these broader questions of audience emerged more quickly and were primarily pursued by the non-technical participants, unlike Group 1. Rather than beginning with the practical relationship between programmer and computer, they more directly questioned who code is for, what it means to understand code, and what kind relationship is established between code and its reader.

> **Gloria:** Clearly [the act of reading out loud and analysing] is an example of code not being purely for the computer [...] in order to make this functionality happen it is also necessary to communicate with oneself and other programmers who may interact with this certain code. So there is a continual multifaceted interaction happening, which is extremely connected to meaning making and understanding as well.

This corresponds with the broader tendency of Group 2 to politicise the material more readily, moving from the particular code fragment toward questions about its social and institutional context.

Broader Commentary

Broader commentary refers to moments where participants moved beyond the immediate code text to consider what it might reveal about wider social, cultural, or conceptual conditions. The most common initiators of these conversations were the two most humanities-oriented participants in the respective groups, Laura and Gloria. Beyond these two participants, however, there was little in indication of a clear discernible pattern in who produced these comments. In Group 1, Laura was followed Ada and Marvin, and then Alan and Charles, in order of frequency with initiating a reflection on broader commentary. In Group 2, Michael was close behind Gloria in

frequency, then farther behind was Jacques. Claude did not contribute any comments to this category.

The second code text generated considerable more broader commentary in both groups. The first, more decontextualised code, tended to operate more as a novelty, generating discussion around its (potentially surface-level) aesthetics, visual appearance, and formal strangeness. There were exceptions, particularly the discussions of audience described above, but these were comparatively uncommon. The second text, by contrast, provided participants with a much more recognisable social and institutional context from which broader commentary could emerge. Its relationship to the European Union, surveillance, data collection, and institutional practice therefore provided rich material for participants to move form the code itself toward questions about the world in which code operates. In the end, we see that Group 1 produced 7 remarks pertaining to broader issues out of the first code text and 14 from the second. Group 2 produced 9 from the first code text and 13 from the second, meaning a higher proportion of Group 2 annotated remarks were represented here.

Computer Science and Software Engineering Culture

Both groups discussed disciplinary cultural conventions, though despite appearing at a similar rate (16 coded remarks from Group 1 and 15 from Group 2), the topic represents a higher proportion of Group 2's total coded comments. These discussions were germinated in similar ways. A participant would offer an interpretation, after which someone with technical knowledge would contextualise it through a particular convention of programming or computer science. Usually, this technical information was offered relatively early in the discussion and group subsequently moved on to consider implications.

There were, however, important difference in how participants positioned themselves in relation to these conventions. In Group 1, participants sometimes explicitly situated their own interpretive difficulties within what they understood as a process of cultural conditioning. This relates to the previously discussed moments in which participants described feeling unable to see the poetics of code because they had been trained to read it primarily in technical terms. It also appeared in reflections on their own programming practices, preferences, and experiences, where participants considered how their individual approaches had been shaped by broader conventions within computer science and engineering.

Group 2 more commonly treated disciplinary and industry conventions as an external explanatory framework. In particular, Claude frequently invoked convention to redirect alternative interpretations, suggesting that a reading resulted from participants' unfamiliarity with established programming practices. We have seen a number of these instances in other sections such as the quote from Claude in section 5.2.2, *Definitive Authoritative Figures*. Michael was more willing to consider the possibility that these conventions could themselves be subverted or instrumentalised to produce poetic effects. At the same time, Jacques and Gloria pushed further against the

idea that an artefact's position within a particular technical culture should exempt it from critical inspection. Rather, they treated these conventions themselves as objects of analysis or as contextualisations to deeper poetic examination of the code.

> **Jacques:** I think this is very normal, that we are reading with certain prejudices or pre-knowings of what [the code] is, unlike a computer who's just reading this and performing a function. As the human, we can read with the already existing knowledge we have and from that lens. And, for example, it's also different when a programmer reads it. I guess [there] can also be desensitization to it, but when you start reading it just by knowing like the cultural meaning of such things you can spot them easily like what this could be used for.
>
> **Claude:** Yeah, no, that's true. I do agree with that.
>
> **Gloria:** Yeah. I think saying that words just are just [words] isn't possible; "I think that a word just is". Just like I think when you use the term desensitize, I think that is kind of a culture that is then generated in that we're just using these words because it's easy to function but they're *words* and they're words with meaning and they are inherently interconnected with meaning and you cannot remove that [meaning]. You can't make it an objective state. So I think obviously, maybe as an outsider perspective, I might be bringing like a certain more critical lens, but that doesn't mean if this is imbuing me with this kind of signification and this knowledge that there's no truth [to it]. There's no ultimate truth to what something is.

This distinction is significant. In Group 1, computer science and engineering culture was often something participants used to reflect upon their own interpretive conditioning. In Group 2, it more frequently became an object of critique in its own right, like what we see in the discussion between Jacques, Claude, and Gloria.

Positionality and Context

Positionality and context differs form the broader discussion of software culture in that it concerns the specific conditions in which the code exists rather than computer science and engineering as a broader social formation. In the first code text, this meant the context of the International Obfuscated C Code Competition and its particular culture of programming, while in the second it meant the European Union, the groups and individuals who were commissioned to develop and maintain the code, and the institutional purposes within which the code operates. Where the previous category concerns *computer science and engineering*, this concerns where and why a particular instance of computer science and engineering is being done.

These discussion occurred overwhelmingly around the second code text. The few examples associated with the first tended to concern audience in a

more abstract sense rather than the situated context of the code itself. This again reflects the different affordances of the two texts; the first offered participants a relatively decontextualised technical artefact, whereas the second provided an explicit institutional and political context from which to interpret code.

Interestingly, the disciplinary distribution of these discussions differed between groups. In Group 1, technical participants were more likely to initiate contextualisation, although their contextualisation tended to remain relatively close to the structure of the code itself. Participants considered, for example, how particular classes, objects, and methods related to the wider codebase:

> **Alan:** I was thinking that okay this is returning this, that collector output, but what structure does it have? Is it an object? Like, what other methods and fields does it have? How can you use it? Like collector types.js and inspect [types]?
>
> **Charles:** Yeah, yeah, exactly. I have to find that.
>
> **Alan:** Both that and like where [are you] running this code class. What...like how what are you using [it]? Like how is it being logged? Is it like using these collector adaptations? output.cookies somewhere else too or, like, how is it being used, this code. We know what it does but how is it being used?
>
> **Charles:** That's the question: for who calls the inspector? That's not something that we can necessarily derive from this because, much like art, you make it and put it out there. How people consume it is up to them. But I know they didn't intend [for] us [to be] sitting here.
>
> **Alan:** I don't think this is being used in more than one file. I probably guess that this is being used in one instance. Like there's an instance of like inspector and it's being ran in another file so it would've been nice to see that also to like get a better [picture].
>
> **Charles:** Part of a larger project you said? How big is said project?
>
> **Facilitator:** I don't know off the top of my head.
>
> **Charles:** Mildly enormous...

This relational understanding could then became a source of poetic interpretations, including comparisons between the relationships of classes and objects and the relationship between poems within an anthology.

In Group 2, contextualisation moved much farther outward. The humanities participants more readily situated the code within broader political and economic structures, including the European Union and capitalism. We can see that in many of the quotes seen thus far from Jacques and Gloria about the loaded nature of the words being used, but also when Gloria says:

> **Gloria:** Do you do you think that maybe could be this decision [to name variables in this way] could be influenced by this kind of official institute? Like the European Union for example; it being too like casual almost or informal to shorten everything?

And also:

> **Gloria:** I think that this is essentially quite a politicized code in a way, because, I mean, it's by the European Union; they're collecting and monitoring.

Comments like these are consistent with the group's tendency toward political investigation, where the code was frequently treated as an artefact embedded within institutional and ideological relations rather than simply as an isolated technical object.

Value Embeddings

This final category concerns moments where participants treated the code as containing or expressing values beyond its immediate technical function. This positions it tightly within a nexus of authorial intent, positionality and context, and software culture. However, the frequent reoccurrence of this specific formulation of these three categories together prompted us to acknowledge a new category for such comments.

These instances occurred exclusively in relation to the second code text and was strong example of epistemic re-orientation among the technical participants in Group 1. Here, participants began to make broader, non-teleological claims about what the code revealed about the values, priorities, or world-view of the people and institutions involved in producing it. Rather than simply asking what a particular implementation does, they considered what its language, structure, and organisation might reveal about what its authors consider important.

Technical participants took the knowledge they had previously been using to explain how the code worked and began using it as the basis for considering what the code might value or assume.

> **Marvin:** But also, Florian Krämer talks about this specifically, like the cultural texture layer that if we, well I kind of write about this, if we take these ideas that are built into the code itself [...] it's really only for us. It's only the the culture that should care about it. Computers doesn't care about it. Or shouldn't. Does it need to care about it?

In this sense, the technical and contextual dimensions of the code were not discarded in favour of political interpretation, but become the material through which that interpretation was constructed.

The dynamic was somewhat different in Group 2. Here, value embeddings were more frequently introduced by Jacques and Gloria, the humanities oriented participants, and directed toward Claude, who was resistant to

these readings. Michael was comparatively receptive. The resulting conversations therefore often took the form of an argument over whether technical language and implementation could legitimately be read as expressions of values, rather than simply as function or conventional decisions. Previously discussed quotes from Gloria and Jacques in the recent sections demonstrate these discussions well.

Summary

Contextual and political reflection most often emerged when participants moved from the immediate code fragment toward questions of audience, disciplinary culture, and institutional setting. Discussions of audience began with who code is written for and often exposed tensions around readability, expertise, and interpretation. Group 1 initially approached these questions more technically, especially through the contrast between human and machine readers, while Group 2 moved more quickly toward social and political questions of who is expected to understand code and whose perspective is privileged. Broader commentary followed a similar pattern: the second code text generated considerably more discussion of surveillance, data collection, the European Union, and institutional practice than the comparatively decontextualised Adder text.

Computer science and software engineering culture also became an object of analysis in both groups. In Group 1, technical participants often reflected on their own disciplinary conditioning and how programming conventions shaped what they were able to perceive in the code. In Group 2, these conventions were more often treated as something to critique or contest, particularly when Claude invoked them to explain away alternative readings and Gloria or Jacques argued that convention itself remained culturally and politically meaningful. Positionality followed the same divergence: Group 1 tended to contextualise the code in relation to its wider codebase and technical ecosystem, while Group 2 situated it more readily within institutional and political structures such as the European Union, capitalism, surveillance, and data governance.

The strongest form of this contextual movement appeared in discussions of value embeddings. These occurred exclusively around the Website Evidence Collector, where participants began treating naming, structure, implementation, and technical organisation as possible expressions of values and priorities rather than as neutral engineering decisions. In Group 1, this was especially significant because technically trained participants began using their functional understanding of the code as a basis for broader cultural or political interpretation. In Group 2, similar claims were introduced more often by Gloria and Jacques and became sites of disagreement over whether technical language could legitimately be read as ideological or value-laden.

5.6 Epistemic Shifts

The final area of analysis concerns the moments in which participants appeared to reconfigure their understanding of code itself. These reconfigurations were not always explicit. Some appeared directly, through participants stating that their understanding of code, poetry, aesthetics, or interpretation had changed. Others appeared more indirectly, through the questions participants began to ask, the analogies they reached for, the kinds of examples they introduced, or the willingness with which they returned to poetic and non-teleological readings over time. For this reason, this section does not treat epistemic adjustment only as a declared change in opinion, but as a shift in the participant's mode of attention.

One very prolific indicator that did emerge was the increasing rate of direct comparison and contrast between code and traditional forms of poetry. In Group 1, the majority of explicit comparisons to poetry occurred during the discussion of the second code text, the more conventionally structured code. This is significant because it suggests that poetic comparison was not restricted to the visibly strange or overtly aesthetic aspects code, more popular in code text 1. While *Adder* more immediately invited comparison through visual form, obfuscation, and authorial play, the second text encouraged participants to locate poetic qualities in more ordinary technical structures: repetition, rhythm, hierarchy, readability, and functional elegance. The more "normal" the code appeared, the more significant it became that participants could still read it poetically.

This is not to say *Adder* did not produce comparisons. Marvin directly compares the visual nature of the code to "concrete poetry", as mentioned before. Alan compares it to ASCII art, to which Marvin pushes back claiming it is trying to be more poetry than any form of illustration. Ada made the comparison that how an observer relates to an interpretive object is similar in poetry and code;

> **Ada:** I also came across a video on YouTube about what is poetry. And it was talking about how different kinds of poetry exist, like visual poetry and then the words and different types, I don't remember. So, I could relate to this [code], like, if you look it, it [is] kind of a visual poetry as well. And then recently I came across a dialogue in a series. It said, like, poetry is not something to be actually *studied*. It is something to be lived, engaged, and felt.
>
> **Facilitator:** And how do you [...] feel about that sentiment actually? Like with regard to [this code]? I mean if poetry is something to be experienced, how do you see that relating to this? You brought it up, so I suppose in some way you've made some connection, but I'm curious if you want to follow that line further. [...] Also if you don't want to say any more [you don't have to].
>
> **Ada:** (Jokingly) Obviously that's no! No, if we look at the engagement part, like we all did do like engagement with this kind

> of stuff, so it was something related...like yes, we did engage with it. Although in different ways, but yeah, we all felt differently.

Ada clearly sees that the ways poetry and the presented code make people feel and think, but most importantly, *engage*, are similar. Like this example, Group 1's comparisons drew on personal encounters with poetry, but also different cultural traditions and poetic categories. Beyond concrete poetry, connection to epic poetry, oral traditions, satire, constraint-based writing, and other forms were also mentioned. This variety was distributed across much of the group, with particularly frequent contributions from Laura and Marvin, as well as several from Ada and Alan. Charles contributed the least from this category, which may partly reflect his limited prior exposure to poetry compared to the rest of the group. The range of comparisons in Group 1 suggests that poetic reading became a shared exploratory practice rather than a specialised contribution from only the humanities participants.

Group 2 displayed a different pattern. In total, they simply made less comparisons (making only 27.59% of the total annotated remarks). But beyond that, their discussions were less likely to compare code to specific poetic techniques of genres, and more likely to move toward contextual, institutional, or political interpretation. Gloria made almost all of the explicit comparisons, 75% to be exact. Michael made the remaining 25% of comparisons. We hypothesise this may be another factor influenced to some degree by social environment, seeing as we can observe that Gloria made half of her comments in the first session, and fewer each subsequent session, making only 1 in the final meeting. It seems that lack of engagement and the interpretive tensions may have discouraged such statements. The remarks Gloria did make were of the most explicit and direct of the whole sample group;

> **Gloria:** I think I see a similarity between this form of code that you guys are talking about and then [the] kind of form that poetry and literary, so poetic form, in [that] you have, for example, free form which is you can basically write down however you would like, and then you have for example rules like a sonnet or like a haiku and then you have to follow these strict guidelines of this form that you have to take. So what you guys use, that you have like a certain space where you have to start when you're writing with, I don't know, the Python or the C code or something. And then if you choose a different form of code, then you get the freedom to, as in this for this example, they get the freedom to like move more in (referring to the indentation) [...] I think there's freedom of usage of space then and that's very similar to form and poetry.

The second major indicator of reconfiguration was epistemic adjustment. Explicit adjustments were relatively easy to identify: participants would state that they had begun to see code differently, or that their understanding

of poetics, aesthetics, or technical form had shifted. Implicit adjustments were more difficult to trace, but they were visible in the development of participants' questions and interpretive habits. In Group 1, this appeared through increasingly thoughtful questions directed both to the facilitator and to one another. Participants became more willing to ask what counted as poetic, whether function could be aesthetic, how poetry might be defined, and how code might be read if it were not approached only as a tool for execution. This curiosity extended beyond the sessions themselves, as several participants discussed poetry outside the formal setting, shared poems with one another, and returned to the sessions with a more developed vocabulary for poetic comparison.

In Group 1, epistemic adjustments appeared most clearly among Alan, Charles, and Ada. For Alan in particular, the shift was not from rejection to full poetic openness, but from technical immediacy toward a slower recognition that poetic readings could make sense even when they did not originate from technical understanding. The comments he made that demonstrated a shift in epistemic framing ranged from direct, explicit comments;

> **Alan:** ...code can be poetic. And [poetic] has a different meaning with for 'poetic code'; like it's not necessarily about rhyme or...it's about creativity in how it looks, what it does, sarcasm in it.

To more subtle demonstrations that the way he is thinking is reorganising such as his increased frequency in asking questions about affect and associations, the engaging in non-technical conversations, and even the general increase in enthusiasm with participating in reading out loud and offering his thoughts. His reflections suggest that collective close reading allowed him to notice how others were perceiving the same code differently, and that this difference became analytically productive rather than simply incorrect or irrelevant. Laura and Marvin showed fewer explicit adjustments, but this seems less a sign of resistance than of prior openness. They entered the sessions already more comfortable with non-teleological and poetic modes of interpretation, so their shifts were less dramatic.

Group 2 showed a somewhat inverse pattern. Here, the most notable epistemic adjustments came not only from technically trained participants, but also from the media and humanities-oriented participants. Gloria and Jacques described changes not simply in their understanding of code, but in their understanding of aesthetics and poetics. As was discussed in section 5.4, they began to consider how, in a coding context, aesthetic value might be tied to qualities such as efficiency, solution, complexity, readability, or elegance. This is important because it shows that the close reading method did not only ask technical participants to become more poetic. It also asked non-technical participants to revise their understanding of poetry and aesthetics in relation to technical constraint.

Across both groups, epistemic adjustment increased over the course of the sessions. This does not mean that all participants arrived at the same understanding of code, or that technical and poetic interpretation were fully

reconciled. The shifts were uneven, partial, and shaped by the social conditions of each group. Nevertheless, the pattern suggests that repeated collective close reading changed what participants were willing or able to notice. Code became less readily reducible to instruction, execution, or correctness. It became available as a textual, aesthetic, affective, cultural, and situated artefact.

The significance of these reconfigurations lies in their reciprocity. Technical participants were not simply taught to see code poetically, and non-technical participants were not simply given technical access. Rather, both groups encountered pressure on their existing categories. For some, code became more poetic than expected. For others, poetry became more functional, constrained, or procedural than expected. This reciprocal adjustment is central to the effectiveness of collective close reading as a method. Its value was not that it produced a unified definition of code poetry, but that it disturbed the assumed separation between code as technical object and poetry as aesthetic object. Through this disturbance, participants began to understand code as a form whose meanings are produced not only by what it does, but by how it is written, read, named, structured, and situated.

5.6.1 Summary

Explicit comparisons between code and poetry were strongly uneven between the two groups. Group 2 produced only 27.59% of all annotated poetry-comparison remarks, with 75% of those coming from Gloria and the remaining 25% from Michael. Group 1 therefore accounted for nearly three quarters of the comparisons, which were also distributed across more participants and drew on a broader range of references, including concrete poetry, ASCII art, epic and oral poetry, satire, and constraint-based writing. In Group 1, most of these comparisons occurred while reading the second, conventionally structured code text rather than the overtly aesthetic Adder. This suggests that, over the sessions, participants became increasingly capable of locating poetic qualities in ordinary programming structures such as repetition, hierarchy, readability, and functional organisation rather than relying on obvious visual resemblance to poetry.

The epistemic adjustments were also uneven between participants but occurred in both disciplinary directions. In Group 1, the clearest changes appeared among the technically experienced Alan, Charles, and Ada; Alan, for example, moved from describing his technical familiarity as inhibiting alternative associations to explicitly stating that "code can be poetic" and increasingly participating in affective and non-technical interpretation. Laura and Marvin showed fewer explicit adjustments, largely because they entered the sessions already receptive to these modes of reading. In Group 2, the shift was particularly visible in Gloria and Jacques, whose understanding of poetics expanded to include technical qualities such as efficiency, complexity, readability, constraint, and elegance. The observed epistemic shift was therefore reciprocal: technical participants increasingly recognised non-functional dimensions of code, while humanities-oriented participants in-

creasingly incorporated functional and technical properties into what they considered aesthetically or poetically meaningful.

6 Discussion

This chapter returns the empirical findings to the theoretical and methodological propositions that structured the study. Rather than treating the results as a straightforward confirmation or rejection of the anticipated epistemic shift, we consider where the proposed indicators were supported, where they require qualification, and how their interaction complicates the initial distinction between teleological and poetic modes of understanding. From this assessment, we then return to the research questions and consider what collective close reading ultimately made possible as a method for engaging with code.

6.1 Assessing Indicators

The four indicators established in the methodology provide the primary structure through which the effectiveness of collective close reading can be assessed. Each is considered here in relation to the behaviours observed across both groups, with particular attention to differences between the anticipated form of the indicator and the manner in which it materialised during the sessions. Their assessment is therefore not simply binary; confirmation frequently arrives alongside complication, contingency, or an unexpected reconfiguration of the phenomenon being measured.

6.1.1 Reconfigurations of How Code Is Understood.

Both groups indeed demonstrated movement from instrumental to expressive commentary. However not to the same degrees and not in a unidirectional capacity. While we definitely saw the introduction of traditional poetic forms of analysis, we also saw a maintenance of technical evaluation; correctness, efficiency, and function remain centre stage, however these features themselves often became important dimensions for poetic analysis. This nevertheless supports a reconfiguration of how code is understood, albeit in a more entangled, though perhaps more interesting, manner than initially anticipated. We also saw the previously predicted centring of politics and authorship. Finally, the proposed meta-statements ("I never thought of code in this way", for example), as we saw, also appeared. Usually it was in the form of a direct comparison to poetry, which was a method we did not expect to see as much, perhaps grounding the epistemic reevaluations with an epistemic anchor made the concepts easier to communicate. Nevertheless, we did indeed see, in both groups, moments where technical participants remarked at how they saw this new, previously unnoticed perspective on code. Particularly in the exit interviews, as predicted, where nearly all technically experienced participants admitted to this reconfiguration, usually with great enthusiasm (the only exception was Charles who was unable to attend the exit interview due to personal reasons).

However, there was noticeable disequilibrium between the amount each group engaged in these transformations. Group 1 produced far more examples of meta-statements and direct expressive engagement, whereas Group 2 dominated the political and positional analysis of the code texts. It is clear that close reading as method does not precipitate the same forms of epistemic reconfigurations in every group, it is contingent on a number of factors.

We observed three noticeable factors: the social dynamics of the groups, individual personalities of the participants, and the manner in which the groups were formed. We suspect that the noticeably heightened atmosphere of positive cohesion in Group 1 likely played an influential role in inviting the technical participants to be more vulnerable in there disciplinary adventurousness. Since we observed better fraternisation as a group, in contrast to Group 2, we speculate they felt more charitable towards the ideas being shared by the non-technical participants. On the second point, the non-technical participants seemed less politically engaged. In Group 2 it was clear the literary and media studies participants were more politically engaged, being quicker to move to this point, contributing to, what may have been perceived by Claude and Michael as, a disciplinary hostility.

Furthermore, the technical participants in Group 1 seemed more predisposed to an interest in poetics and more readily accepting of alternative conceptualisations of code. The reason for this focus of more accepting technical student may well be due the final factor; the manner of group formation. With the first group being formed via open-call, it may have attracted participants already somewhat, perhaps subconsciously, willing to have their views on technical epistemology challenged.

The final, unpredicted point that further contributed to the confirmation of this indicator was the converse; we saw media and literary participants, namely Jacques and Gloria, mention they perceived a shift in what they saw as poetic. We expected these participants to accept an 'alternative' attitude to the potentiality of code being poetic, but not what constitutes as poetic. This was a very interesting discovering and has implications in where this sort of practice and exercise could be applied.

Taken together, the findings confirm this indicator, although in a more entangled form than initially anticipated. The movement was not from instrumental to expressive understandings so much as an expansion of what could be understood as meaningful within the technical. Technical considerations of function, efficiency, and structure remained present, but increasingly became themselves available for poetic and expressive interpretation. The reconfiguration was therefore not a displacement of technical understanding, but a broadening of it.

6.1.2 Expansion of Interpretive of Authority.

At the beginning of the sessions, interpretive authority was visibly concentrated among the technically oriented participants. Technical participants were not only more likely to speak first and more frequently, but also tended to establish the initial interpretive frame through which the code was

approached. In both groups, a relatively clear disciplinary representative emerged: Alan in Group 1 and Claude in Group 2. Over the course of the three sessions, however, this concentration began to loosen, with a greater diversity of participants contributing to the direction and development of interpretation. This suggests an initial movement toward a more distributed interpretive authority, although, as the following discussion demonstrates, this did not amount to a straightforward displacement of technical expertise.

However, at no point did this behaviour of offering technical perspectives evaporate, and yet, we still see a change in disciplinary hierarchy. In session 1, technical explication was exclusively in service of explaining, definitively, what the group was reading, yet, by session 3, technical perspectives were still being offered (and asked for) but now in service of deeper poetic or political analysis. When we review this indicator concurrently with the previous indicator, what we notice is that technical authority developed a new facet, one of poetic authority. It was not that by session 3 technical explanations were unnecessary or epistemically inhibiting to poetics, it was that technical explanations were simply a particular dimension to be used for poetics. Therefore we must see that not only did the volume of explanations dwindle and the proportion of technical spokespersons attenuate, but also the purpose of the technical explanations shifted and became inverted (at times) to serve the poetic perspective. Both of these aspects were consistent across groups.

When trying to understand whether the non-technical participants felt more empowered to speak direct analysis becomes slightly more complicated. In Group 1, the non-technical participants were actually most defensive and outspoken in the first session. One way to interpret this is that they stopped explicitly defending their perspectives as the sessions progressed, however what we must consider is that the need to make explicit comments decreased. When we consider the noticeable increase in acceptance of alternative positions and ideas, it would seem to make sense that the number of annotations from a literary or media studies participant fighting over a claim would reduce. In Group 2, the image is even more complicated. Tensions over the validity of interpretations were present throughout all sessions; both visible in the field notes and the transcription. It was clear, through subtle gestures of not responding to claims, being first to speak, and not elaborating or investigating alternative positions that hierarchy in Group 2 was maintained. However, we must also consider that in the exit interviews the participants from technical backgrounds did admit that they appreciated the alternative perspectives. Furthermore, in this Group, there was indeed an increase in non-technical conversation domination and defence. Therefore, what we observe is while the technical participants didn't engage *more*, the non-technical participants definitely felt more compelled to offer their perspectives.

This suggests that close reading as an exercise will not necessarily redistribute interpretive authority explicitly nor equally. We can see changes and transitions within the authoritative landscape. Interpretive authority was distributed differently; it was not a straightforward de-monopolisation

of technical expertise as suggested, rather, the sessions demonstrated a redistribution in the *function* of authority. Technical knowledge remained authoritative but increasingly operated as one resource within a broader interpretive practice. The manner in which the hierarchy of disciplines and perspectives materialised also remained individuated and contested. Furthermore, the extent to which this transformation occurred remained contingent on the social and disciplinary configuration of the group. Therefore, the second indicator is, like the first, confirmed with qualifications.

6.1.3 Sustained Speculative, Non-Teleological Discourse.

This indicator was considerably more divisive between groups, and mirrored many of the differences already observed across the preceding indicators. In Group 1, the technical participants became noticeably more comfortable with each discussed form of non-teleological investigation over the course of analysis, especially in the case of code text 1. Self-reflective comments from participants such as Alan, together with their later reflections in the exit interviews, suggested that Group 1 gradually developed an environment conducive to associative and speculative interpretation. Non-teleological readings were not only produced with increasing frequency when reading a text, but they were increasingly allowed to remain in discussion without requiring immediate technical resolution.

Group 2 presents a more complicated case. Claude did not initiate any associative interpretation or speculative postulations into the author, common occurrences from his session colleagues. He was frequently the most likely to redirect speculative discussion toward technical explanation, programming convention, or missing contextual knowledge. Michael engaged considerable more with associative and speculative thinking, though not to the same extent witnessed among the technical participants of Group 1. Importantly, however, Jacques and Gloria continued to produce and sustain on-teleological interpretations throughout the sessions. Group 2 therefore demonstrates that collective close reading can generate an environment in which speculative discourse occurs without necessarily producing equivalent participation in that discourse from every member. The method provided opportunities for association and speculative interpretation, but participation in these opportunities remained contingent upon the interpersonal and disciplinary dynamic of the group.

This distinction between participation in speculative discourse and appreciation of it is further complicated by the exit interviews. Even participants who most frequently redirected discussions toward teleological certainty described the speculative conversations as stimulating and memorable. In some cases even impactful beyond the scope of the sessions and made them reconsider code going forward. Claude himself recalled interpretations offered by the non-technical participants that had remained with him after the sessions. Collective close reading therefore did not necessarily compel participants to perform speculative openness during the discussion, even where they later recognised value in the interpretation produced through it. The creation of an interpretive opportunity should not, then,

be confused with the uniform adoption of the interpretive behaviour the method attempts to facilitate.

One predicted quality that appeared considerably less was the sustained coexistence of competing or disjointed interpretations. Participants were generally comfortable entering into uncertain and unverifiable discussions, but these conversation nevertheless tended to move toward some form of consensus. The persistence of speculation therefore did not necessarily entail the persistence of interpretive plurality. Audibility and the aloud recital of code provided one notable exception, however, generally, the groups appeared inclined to collectively settle upon a particular speculative trajectory rather than indefinitely sustain several possibilities simultaneously. This tendency was not confined to technically trained participants, but appeared among those with humanities and media backgrounds as well.

What this indicator demonstrates, then, is that collective close reading can produce conditions in which non-teleological and speculative interpretation emerge and are sustained, but it does not guarantee either equal participation or enduring interpretive plurality. Its effectiveness appears sensitive to the social configuration of the group, the dispositions and disciplinary habits of the participants, and the particular interpretive affordances of the code text under interpretation. Most importantly, the method appears capable of delaying the return to teleological resolution long enough for associative, speculative, political, and authorial interpretations to develop. The indicator can therefore be considered confirmed with qualification; sustained non-teleological discourse clearly emerged in both groups, but its distribution was uneven, socially contingent, and more frequently oriented toward eventual consensus than the coexistence of multiple readings.

6.1.4 Consistent Emergence of Common Poetic Features

Of the proposed indicators, this was perhaps the most straightforwardly confirmed. A substantial portion of the analysis was ultimately dedicated to the cataloguing and discussion of the poetic categories and features that repeatedly emerged through the collective close reading as a practice. Participants identified sensory and affective qualities, textual and linguistic affordances, structural and compositional forms, and contextual and political dimensions across both texts. More importantly, these observations were not limited to isolated or idiosyncratic associations. Different participants, often across disciplinary backgrounds and between groups repeatedly returned to comparable features such as naming, repetition, rhythm, structure, visual arrangements, audience, and the contextual values carried by technical decisions.

This recurrence lends support to the proposition underlying the indicator. While we cannot empirically demonstrate poetic qualities existing autonomously, independent of a reader, this is beyond the scope of our working definition of poetics. What we observe is code possessing recurring affordances which are available as poetic material when our mode of attention is changed and when we dwell within this disciplinary interstice. What is the most compelling aspect is that despite situated and, at times,

idiosyncratic instantiations of poetic features, the sites around which they accumulated were noticeably recurrent.

The distinction between features and forms was consequently also productive. Individual observations can be understood as particular instantiations while their recurrence allowed them to become a feature. We can then gather these into broader poetic categories, resulting in a taxonomy of poetic forms. However, this is of course not an exhaustive grammar of code poetics, it is an empirically produced vocabulary for describing the kinds of poetic affordances that became legible through the sessions.

This indicator can therefore be considered confirmed. Perhaps more significantly, its confirmation suggests that poetic engagement with code was not sustained solely by participants importing an external library vocabulary onto an otherwise inert technical artefact. Certain technical, textual, structural, and functional qualities repeatedly became sites of poetic attention across readers. What emerged was less a fixed poetic hidden within code than a recurrent field of poetic potential made perceptible through the particular practice of collective close reading.

6.2 Research Questions

Returning, then, to the primary research question: *What is the effectiveness of collective close reading as a method for engaging with code texts in producing an epistemic shift from teleological interpretation to poetic modes of understanding?* We find the method to be effective, though not entirely in the form initially anticipated. As established earlier, "effectiveness" here does not refer to improvement against a control condition, but to identifiable qualitative shifts in discourse, interpretive authority, and epistemic framing. Across both groups these shifts were indeed present. However, what emerged was not a clean departure from teleological interpretation toward some independently poetic register. Technicality persisted. Function, correctness, efficiency, convention, and technical explanation remained present throughout every session. What changed was the epistemic monopoly. Technical aspects ceased, at least intermittently, to operate as the terminus of interpretation and instead became dimensions among others through which code could be understood.

This distinction is important. The proposed shift should not be understood as a movement from one stable epistemology into another, as though technical understanding were abandoned and replaced by poetic understanding. Rather, we witnessed an expansion of the field through which code became intelligible. A technically necessary structure could remain technically necessary while also becoming compositional; a variable name could maintain its functional referent while accumulating affective or metaphorical associations; efficiency could remain an engineering concern while simultaneously becoming elegant or expressive; a piece of institutional software could continue to perform its intense operation while also becoming politically and culturally radiant. This relation between technical and poetic interpretation became one of the more unexpected findings of the study. The poetic

did not appear only when function was suspended. Often, it appeared when function itself became available to a different mode of attention.

The three indicators used to approach the primary research question therefore cannot be understood entirely independently. The reconfiguration of how code was understood, the expansion of interpretive authority, and the sustaining of speculative, non-teleological discourse appear instead as different dimensions of the same epistemic movement. As technical authority became less capable of immediately exhausting interpretation, speculative and associative readings could persist for longer. As these readings persisted, participants encountered dimensions of code that had previously remained peripheral or illegible. Repeated exposure to these dimensions could, in turn, alter what participants considered relevant to understanding code in the first place. What we observe is less a procession of discrete effects than a recursive movement; interpretive authority changes the kinds of discourses possible, those discourses change what becomes perceptible, and these newly perceptible qualities further destabilise the previous hierarchy of interpretation.

This is particularly consistent with the working definition of poetics developed earlier in this research. Poetics was proposed not as a genre, ornament, or a classification of particular artefacts, but as a mode of attention; a relational operation in which meaning is realised through sustained engagement rather than stabilised through reduction. The primary object of analysis was therefore never the isolation of objectively poetic properties within code, but the reader-code relation under a poetic frame, with poetic features operating as empirical traces of that relation. The code texts did not change, nor did the participants cease to understand their executable or instrumental qualities. What changed was the relations through which those qualities were encountered. They were enlarged.

This enlargement was not uniform. Group 1 and Group 2 demonstrate quite clearly that collective close reading does not function mechanistically. The method does not simply produce an epistemic shift upon application. The social conditions of the group, disciplinary backgrounds, individual dispositions, the manner in which the participants were brought together, and the particular affordance of the code text all mediated what was able to germinate within the sessions. The collective element is therefore not simply a container around the method, but instead is constitutive of the interpretations themselves. Who speaks first, whose interpretations receive credibility, whether disagreement is explored or resolved, and how speculative claims are received all shape what can subsequently become visible in the code. Collective close reading appears capable of opening an interpretive space, but it cannot command what participant will do within that space.

This contingency does not, however, signify a failure of the method. Even Group 2, whose technical hierarchy remained more pronounced and speculative discussion more frequently redirected, produced sustained non-teleological discourse, political and authorial interpretation, altered distributions of participation, and retrospective acknowledgment from technically oriented participants that alternative readings had become memorable or

significant. Group 1 demonstrated these movements more visibly and more frequently, but the difference between groups is itself instructive. Effectiveness appears not as the production of one predictable end-state, but as the capacity to disturb an established interpretive relation sufficiently for other forms of knowing to become possible.

There is also an important bilateral quality to this reconfiguration. We initially anticipated that most visible epistemic movement would come from technically experienced participants becoming receptive to poetic modes of understanding. This occurred, but it was accompanied by the converse. Participants from literary and media background also began reconsidering what they understood as poetic, particularly when confronted with the aesthetic implications of efficiency, constraint, technical elegance, repetition, or functional organisation. The poetic frame was therefore not simply carried into the sessions and disseminated by humanities participants. Instead, there was a reciprocal exchange, an expansion of the disciplinary intersection.

We therefore answer the research question affirmatively, but with qualification. Collective close reading was effective in producing identifiable epistemic shifts away from an exclusively teleological engagement with code and toward poetic modes of understanding across both groups. This shift was partial, uneven, socially contingent, and did not abolish technical interpretation. Nor should it have. Its effectiveness lay instead in disturbing the sufficiency of teleological explanation and allowing technical, affective, linguistic, political, structural, authorial, and speculative relations to coexist within the interpretive field. The epistemic shift was ultimately not from technical to poetic, but from the technical as sufficient toward the technical as one mode of knowing among others.

7 Conclusion

If a house that has been inhabited is not an inert box, if inhabited space transcends geometry, then, perhaps, code that is inhabited is not inert text, perhaps it transcends its programme. Bachelard sat with architecture, a formal tradition, a practice characterised by restrictions and constraints, and showed that through analysis, through dwelling, it sublimates, it becomes more than its compositional formal techniques. However, he did so alone.

Software production, code, is another field that is demarcated, now by parameters and scope. While we indeed can point to linguistic power laws, similar patterns of abstraction, or relationships with linguistic dispositions, it is still very much an algorithmic discipline. However, what we have seen from the application of close reading, is that it does not require strict and formal training as a poet or philosopher to transcend these traditional perceptions of code.

While indeed it may be true that code has been affixed and engendered as a mathematical monolith, like Bachelard demonstrates, it is in inhabitation that meaning can become enriched. Like Césaire explains, it is in the

active mode of attention, the cohabitation within antinomy that we produce poetics. Over the course of three close reading sessions with each group, as the participants applied this dialectic, we indeed saw that discussions, investigations, and discoveries emerged that facilitated novel and orthogonal understandings of code. New perspectives within our participants and the sapling of a code-poetic vocabulary began to germinate within this adjacent paradigm. Like ʻAlī al-Tahanāwī and al-Qādir Bīdil, the participants did not abandon the geology of the volcano, but instead understood the value of the sound of its spirit. They did not reject the science of its magma, but learned to study the knowledge of its bubbling. Code did not lose its teleology, rather, that same functional dimension became heavier with the weight of new associations and ways of knowing.

Collectively, when placed in an environment that encourages, nurtures, and facilitates alternative lines of questioning, sits with and faces tension, and explores in depth what at first may seem disconnected and aloof, participants were able to experience a poetic epistemic shift. This collective close reading environment did not produce a departure from the technical to the poetic, but from the technical as sufficient; toward the technical as one mode of knowing among others. Efficiency, correctness, optimisation etc. became features to be contrasted and compared, antinomiously, with metre, metaphor, and lineation. However, they went even further, becoming forms and representations *of* those very features, and conversely, for those familiar with poetics, these teleological dimensions became new poetic features to be transposed back to the humanities.

This thesis is limited in scope, and we must be transparent and clear that the findings should not be treated as statistically generalisable, but as situated qualitative findings emerging from these particular groups and texts. This limitation comes from a sample size of only two groups, totalling nine students. The demographic of the participants is also limited. All were university students aged between early twenties to early thirties. The group was also skewed male and also skewed European. The experiment, due to its demand for a long time commitment and slow study meant only two code texts were reviewed, limiting the discussion to only two coding languages: C and TypeScript. The qualitative coding framework was necessarily shaped by the theoretical conception of poetics developed in this thesis and by our own interpretive decisions as researchers; there is no control condition or objective framework to which we can compare and contrast, meaning our own conceptions of poetics and biases to what is worth reporting are implicit within the analysis. These limitations create the boundary conditions of what we are able to claim.

This thesis makes three principal contributions: methodological, conceptual, and empirical. Methodologically, it demonstrates collective close reading as a viable form of engagement with source code outside the dominant practices through which code is ordinarily examined. Code review, debugging, optimisation, and programme comprehension are generally convergent practices which seek to establish what code does, whether it does so correctly, and how its operation might be improved. Collective close reading

instead suspends the demand for immediate resolution, slows engagement, permits speculation, and allows disagreement, association, affect, and contextual reflection to remain analytically productive.

Conceptually, this thesis develops a poetics of code that does not depend upon declaring code to be poetry, nor upon locating a set of intrinsic aesthetic properties that would qualify one code text as poetic and another as not. Poetics has instead been approached as a mode of attention; a relational operation through which meaning is realised by sustained engagement rather than exhausted through reduction. This framing proved particularly consequential in the relation between technical and poetic interpretation. The experiments did not reveal a clean opposition between the two. Technical and poetic interpretation were not therefore necessarily competing layers of analysis but could emerge from the same feature under different, and sometimes simultaneous, modes of attention. One conceptual contribution of the study is therefore the complication of the presumed opposition between teleological and poetic knowledge. The poetic need not begin where the function ends.

Finally, the thesis makes an empirical contribution through the identification of recurrent sites around which poetic attention accumulated. Participants repeatedly returned to instances of poetic features which we have broadly distilled into four main categories; sensory and affect, textual and linguistic affordance, structure and composition, and context and politics. These categories should not be interpreted as an exhaustive grammar of code poetics, nor as observer-independent properties residing autonomously within code. They instead provide an empirically grounded vocabulary for describing the kinds of poetic relations that became legible under the conditions of this study.

These contributions open several directions for future research. Following from the points raised in the introduction and background research, the totality of mathematics' grip on software pedagogy is perhaps unjustified in the face of programming's possession of linguistic, communicative, and interpretive dimensions. While the present study does not demonstrate that literary or poetic approaches produce better programmers, nor was it designed to measure learning outcomes or software quality, it does suggest that practices such as collective close reading can alter what students consider relevant to understanding code. Integrating more literary practices, such as (collective) close reading, into existing pedagogical structures, nurturing alternative forms of analysis (such as the poetic) may produce interesting and productive alternative perspectives within the field of engineering, critical code studies, philosophy of (computer) science, to name a few.

This enrichment is not just limited to the disciplines of computer science and engineering, as we continue to emphasise. The reciprocal epistemic shifts observed in the experiments suggest that technical concepts may themselves provide new resources for literary, artistic, and critical practice. What we witness in the integration of teleological metrics into poetic discourse opens new techniques for both poetic analysis elsewhere as well as (poetic) production. Further exploration of the relationship between

concepts such as correctness and optimisation and the poetic nature of an artefact, in the context of code and beyond, should be explored even further.

Similarly, we believe that this thesis may serve as inspiration for a new class of abstract structures, strategies, and algorithms to be generated with concepts borrowed from literary practices. This thesis has primarily investigated what occurs when code is *read* differently; a subsequent question is whether sustained poetic attention can alter how code is *written*. As discussed earlier in the paper, Exman and Plebe (2016) proposed the '*Metaphor* design pattern' (Exman and Plebe, 2016, p. 14), exemplifying one instance of how concepts derived from literary studies might be translated into software structure. There may be innumerable structures and concepts to transmute and transpose into the software domain. Rather than treating literature merely as a vocabulary for describing completed software, such work could ask whether literary and poetic concepts can become generative constraints within software production itself.

These possibilities should be investigated alongside broader replication of the present study. The small and demographically narrow participant pool, the use of only two code texts and two programming languages, and the socially contingent character of collective interpretation all restrict the degree to which the present findings can be generalised. Future studies could therefore vary programming languages, software domains, levels of technical expertise, group composition, facilitation strategies, and the relationship between individual and collective reading. Such work would allow the recurrent poetic affordances identified here to be tested, contested, expanded, or reorganised, while also clarifying the conditions under which collective close reading most effectively opens code to alternative modes of understanding.

This research has not sought to rescue code from technicality by declaring it poetry, nor to uncover some hidden poetic essence latent beneath its executable surface. Code remained functional, constrained, exact, and machinic throughout; what changed was the relation through which it was encountered. Inhabited differently, the same structures became capable of sustaining associations, tensions, rhythms, politics, affects, and forms of knowledge that teleological attention alone could not exhaust. A poetics of code, then, is not a new classification of software, but an invitation to dwell within it differently. To resist the assumption that what code does is sufficient to describe what code is. A poetics of code begins not by deciding that code is poetry, but by allowing ourselves to ask what else it is. If inhabited space can transcend geometry, perhaps inhabited code can likewise exceed its programme.

References

Adler, M. (1940). *How to read a book*. Simon & Schuster. https://books.google.nl/books?id=2-CEAAAAIAAJ

al Obaidi, Y. (2024). Let Us Meet in the Margins [Barzakh]. In M. Teran, M. Herbst, V. S. Rehberg, R. Turner, & The Promiscuous Care Study Group (Eds.), *Promiscuous Infrastructures: Practicing Care* (pp. 104–121). Journal for Aesthetics & Protest; WdKA Research Center. https://www.joaap.org/press/promiscuous/Promiscuous_Infrastructures.pdf

al-Qādir Bīdil, M. A. (1962–1965). *Kulliyāt* (K. Khalīlī, Ed.; Vols. 3–4). Wizārat-i Farhang.

al-Tahānawī, M. A. (1996). *Kashshāf iṣṭilāḥāt al-funūn wa al-ᶜulūm* (R. al-ᶜAjam et al., Eds.; Vols. 1–2). Dār al-Kutub al-ᶜIlmiyya.

Aristotle. (2008). *Poetics* (S. H. Butcher, Trans.). Project Gutenberg. https://www.gutenberg.org/files/1974/1974-h/1974-h.htm

Bachelard, G. (1961). *La poétique de l'espace* (3rd). Les Presses universitaires de France.

Bachelard, G. (1969, January). *The poetics of space*. Beacon Press (MA).

Barthes, R. (1967). *The death of the author* (R. Howard, Trans.). https://www.d.umn.edu/~cstroupe/handouts/8500/barthes_death.pdf

Bayrami, P., & Rice, J. E. (2021). Code authorship attribution using content-based and non-content-based features. *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1–6. https://doi.org/10.1109/CCECE53047.2021.9569061

Berkenbilt, E. J. (1993). Best obfuscated algorithm: Ejb, puzzles [Accessed: 2026-01-27]. https://www.cise.ufl.edu/~manuel/obfuscate/ejb.hint

Bialostosky, D. (2006). Should college english be close reading? *College English*, *69*(2), 111–116. https://doi.org/10.2307/25472195

Bruder, A. (2020). Slowing down and speaking up. *Transformations the Journal of Inclusive Scholarship and Pedagogy*, *30*(1), 46–47. https://doi.org/10.5325/trajincschped.30.1.0047

Casalnuovo, C., Sagae, K., & Devanbu, P. (2019). Studying the difference between natural and programming language corpora. *Empirical Software Engineering*, *24*(4), 1823–1868. https://doi.org/10.1007/s10664-018-9669-7

Césaire, A. (1990). Poetry and knowledge (A. J. Arnold, Trans.). In A. Césaire (Ed.), *Lyric and dramatic poetry, 1946–82* (pp. xlii–lv). University Press of Virginia.

Chandler, D. (2006, April). Semiotics for beginners: Signs. https://www.cs.princeton.edu/~chazelle/courses/BIB/semio2.htm#:~:text=Saussure%20offered%20a%20'dyadic'%20or,)%20%2D%20the%20concept%20it%20represents.

Chomsky, N. (1956). Three models for the description of language. *IEEE Transactions on Information Theory*, *2*(3), 113–124. https://doi.org/10.1109/tit.1956.1056813

Ciardi, J., & Williams, M. (1975, January). *How does a poem mean?* Houghton Mifflin.

Culler, J. (2015, June). *Theory of the lyric*. Harvard University Press. https://doi.org/10.4159/9780674425781

Dahlstrom, D., & Somayaji, V. (2003, June). Semiotics and uid: Peircian semiotics. https://cseweb.ucsd.edu/~ddahlstr/cse271/peirce.php#:~:text=a%20representamen%2C%20the%20(not%20necessarily,between%20the%20representamen%20and%20object).

Danielewski, M. Z. (2014). Foreword. In M. Jolas (Ed.), *The poetics of space* (pp. vii–xv). Penguin Classics.

Derrida, J. (1997). *Of grammatology* (G. C. Spivak, Trans.). The Johns Hopkins University Press. https://doi.org/10.56021/9781421419954

Downs, H., & Makarios, S. (1995). *Adder* [International Obfuscated C Code Contest]. https://www.cise.ufl.edu/~manuel/obfuscate/heathbar

Exman, I., & Plebe, A. (2016). Software is part poetry, part prose. *Communications in Computer and Information Science*, 365–380. https://doi.org/10.1007/978-3-319-52758-1_20

Fisher, D., & Frey, N. (2012). Close reading in elementary schools. *The Reading Teacher*, *66*(3), 179–188. https://doi.org/10.1002/trtr.01117

Frey, N., & Fisher, D. (2013, January). *Close reading*. Principal Leadership. https://www.mdek12.org/sites/default/files/Offices/Secondary%20Ed/ELA/tools/close-reading-explained.pdf

Gordon, C. S. (2024). The linguistics of programming. *Proceedings of the 2024 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, 162–182. https://doi.org/10.1145/3689492.3689806

Gowen, A., & al Obaidi, Y. (2024, April). Off margin | episode 4: The poetics of protest. https://www.mixcloud.com/radiowormrotterdam/off-margin-5-w-amy-gowen-190424/

Hardin, B. (2024, June). Writing learning into shape: Using concrete poetry to explore & reflect content knowledge. https://literacy6-12.org/writing-learning-into-shape/

Hinchman, K. A., & Moore, D. W. (2013). Close reading: A cautionary interpretation. *Journal of Adolescent & Adult Literacy*, *56*(6), 441–450. https://doi.org/10.1002/jaal.163

Hoover, P., & Chernoff, M. (2004). *New american writing*. Oink! Press. https://books.google.nl/books?id=PFIrAQAAIAAJ

Izzo, D. (2013). Is there a beast in this class?: The pragmatics and ethics of dialogic reading. *The Henry James Review*, *34*(3), 249–260. https://doi.org/10.1353/hjr.2013.0032

Kain, P. (1998). *How to do a close reading*. Harvard University Writing Center. https://www.cablemuseum.org/wp-content/uploads/2016/06/AoW_Harvard_HowtoDoaCloseReading.pdf

Kearney, R. (2014). Introduction. In M. Jolas (Ed.), *The poetics of space* (pp. xvii–xxx). Penguin Classics.

Kingsley, K. S., & Parry, R. (2024). Empedocles. In E. N. Zalta & U. Nodelman (Eds.), *The stanford encyclopedia of philosophy* (Winter 2024). Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/win2024/entries/empedocles/

Kokol, P., & Kokol, T. (1996). Brief communication linguistic laws and computer programs. *Journal of the American Society for Information Science (1986-1998)*, *47*(10), 781.

Louridas, P., Spinellis, D., & Vlachos, V. (2008). Power laws in software. *ACM Trans. Softw. Eng. Methodol.*, *18*(1). https://doi.org/10.1145/1391984.1391986

Mikics, D. (2007, January). *A new handbook of literary terms*. Yale University Press.

Nørmark, K. (2013, July). Lecture notes: Linguistic abstraction. https://homes.cs.aau.dk/~normark/prog3-03/html/notes/languages-book.html#:~:text=The%20concept%20Linguistic%20abstraction:%20Linguistic,functional%20abstraction%20with%20linguistic%20abstraction.

Olsen, S. H. (1976). What is poetics? *The Philosophical Quarterly*, *26*(105), 338–351. https://doi.org/10.2307/2218864

Oxford English Dictionary. (2025, November). *Poetics, n. — definition* [Accessed 02/11/2025]. Oxford University Press. https://www.oed.com/dictionary/poetics_n

Payumi, C. M., & Hartati, Y. F. (2018). The use of close reading technique to improve students reading comprehension. *PROJECT (Professional Journal of English Education)*, *1*(3), 187–194. https://doi.org/10.22460/project.v1i3.p187-194

Pellò, S. (2023). Looking through taḥqīq glasses: Early modern imagination and the unveiling of nature in mīrzā bīdil's the sinai of knowledge. *Journal of Early Modern History*, *27*(4), 368–383. https://doi.org/10.1163/15700658-bja10072

Pierret, D., & Poshyvanyk, D. (2009). An empirical exploration of regularities in open-source software lexicons. *2009 IEEE 17th International Conference on Program Comprehension*, 228–232. https://doi.org/10.1109/ICPC.2009.5090047

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C.-H. (2020). Relating natural language aptitude to individual differences in learning programming languages. *Scientific Reports*, *10*(1). https://doi.org/10.1038/s41598-020-60661-8

Rabinowitz, P. J. (1992). Against close reading. *Pedagogy is politics: Literary theory and close reading*, 230–243.

Tanaka-Ishii, K. (2010). *Semiotics of programming*. Cambridge University Press.

Visser, E. (2015). Understanding software through linguistic abstraction [Special Issue on New Ideas and Emerging Results in Understanding Software]. *Science of Computer Programming*, *97*, 11–16. https://doi.org/10.1016/j.scico.2013.12.001

Zellini, P. (2020, May). *The mathematics of the gods and the algorithms of men: A cultural history*. Simon & Schuster.

Zhang, H. (2009). Discovering power laws in computer programs. *Information Processing & Management*, *45*(4), 477–483. https://doi.org/10.1016/j.ipm.2009.02.001

A Code Selection: Adder

```
#                                                       include <stdio.h>
#                                           define     MAin   printf("%d\n"
#                                           define     mAIN        return 0
#                                           define     MaiN         {static
#                                           define     mAlN          ) {if(
#                                           define     MAlN           char*
#                                           define     MAiN            (!!(
#                                           define     mAiN            atoi
#                                           define     mAln            &1<<
#                                           define     MAlN            !=3)
#                                           define     MAln             )&&
#                                           define     MAIN             int
#                                           define     maln             --,
#                                           define     Maln              <<
#                                           define     MaIn              ++
#                                           define     MalN              |=
#                                           define     MAln              ||
#                                           define     malN              -1
#                                           define     maIN               *
#                                           define     MaIN               =
#                                           define     maIN               )
#                                           define     MaIN               (
#                                           define     Main               ;
#                                           define     mAIn               !
#                                           define     MAIn               }
#                                           define     mAIN               ,
                                            MAIN      mAIn
                                            Ma1N      MAIN
                                            ma1N      mA1N
                                            mAiN      Ma1N
                    MA1N ma1N mA1N maIn MaIN malN mA1N      ma1n
                    mA1N                         maiN      Main
                    MAIN      main Ma1N MAIN Ma1n mA1N MA1N maIN
                    mAin      mAlN               Ma1n      MAlN
                    mAIN      Main               maIn      MaIn
              mA1N Ma1n maln mAin MaIn           Main      maIn
              MaIN                mAiN           Ma1N      Ma1N
              Ma1n                maln           maIN      mAin
              MaIn                ma1N           ma1N      Main
              ma1n                MaIN           mAiN      Ma1N
              Ma1N                Ma1n           maln      maIN
              mAin                MaIn           ma1N      ma1N
              Main                mAIn           Ma1N      mAIn
              Ma1N                mAIn           Ma1N      mAIn
              Ma1N                mAIn      Ma1N mAIn      Ma1N mAIn
               Ma1N              mAIn       Ma1N mAIn      Ma1N mAIn
                 Ma1N          mAIn         Ma1N   mAIn  Ma1N   mAIn
                    Ma1N mAIn Ma1N           mAIn      Ma1N      mAIn
                         Ma1N                Ma1n                ma1N
                         ma1N                ma1N                ma1N
          ma1N ma1N ma1N ma1N                ma1N                ma1N
          ma1N           ma1N                ma1N                ma1N
          ma1N           ma1N                Main                MAin
          mA1N maiN ma1N Main mAIN Main      MAIn                MAIN
          mAIn  Ma1N              MAIN       mAin                ma1N
          MaiN   MAIN            main          MaIN            malN
          Main    main          MaIn             Main        mAIN
          mA1N     maiN        MalN                Ma1N    MAiN
          maIn      mAln      main                     ma1N
          MAIn       Ma1N    MAiN                      ma1n
          mAln        main  ma1N                       MA1n
          mAin           MAln                          Ma1N
          mA1n        MAiN  ma1n                       mAln
          main        MAln  mAin                       ma1N
          ma1N           ma1N      MAln Ma1N mA1n MAiN maIn
          mAln           main      MAln
          Ma1N           MAiN      ma1n
          mAln      main ma1N MA1n mAin MAln
          Ma1N      mA1n                MAiN
          ma1n      mAln                main
          MAln      mAin                ma1N
          ma1N      ma1N                ma1N
          ma1N      ma1N                Maln
          main      mA1N                MAiN
          ma1n      mAln                main
          MAln      mAin                ma1N
          MAIn      MAiN                maIn
          mAln       main              MAln
          Ma1N         MAiN          ma1n
          mAln           main ma1N MA1n
          mAin                MAln
          Ma1N                mA1n
          MAiN                ma1n
          mAln                main
          MAln                mAin
          ma1N                ma1N
          ma1N                ma1N
          Main                MAIn
```

B Code Selection: Website Evidence Collector > Inspector

```
import groupBy from "lodash/groupBy.js";
import {
  RawCollectionData,
  Hosts,
  InspectedCookie,
} from "../collector/types.js";
import { EnrichedCollectionData } from "./types.js";
import { FirstPartyChecker } from "../lib/first-party-checker.js";
import { CookieRecorder } from "../collector/recorder/cookie-recorder.js";
import { BeaconRecorder } from "../collector/recorder/beacon-recorder.js";
import { HostRecorder } from "../collector/recorder/host-recorder.js";
import { WebSocketRecorder } from "../collector/recorder/websocket-recorder.js";
import {
  markFirstPartyCookies,
  flattenCookieEvents,
  mergeCookieSources,
  sortCookiesByExpiration,
  categorizeCookies,
} from "../lib/inspect-cookies.js";
import { LocalStorageRecorder } from "../collector/recorder/local-storage-recorder.js";
import {
  CollectedBeacon,
  EventMetaData,
  InspectedLocalStorageSnapshot,
} from "../collector/recorder/types.js";
import { CookieDB } from "../lib/cookie-db-schema.js";
import { parseUrl } from "../lib/url-utils.js";

class Inspector {
  constructor(
    private collectorOutput: RawCollectionData,
    private firstPartyChecker: FirstPartyChecker,
    private cookieRecorder: CookieRecorder,
    private localStorageRecorder: LocalStorageRecorder,
    private beaconRecorder: BeaconRecorder,
    private hostRecorder: HostRecorder,
    private webSocketRecorder: WebSocketRecorder,
    private cookieDb?: CookieDB | null,
  ) {}

  run(): EnrichedCollectionData {
    const cookies = this.inspectCookies();
    const localStorage = this.inspectLocalStorage();
    const beacons = this.inspectBeacons();

    const beaconHosts = this.hostsFromBeacons(beacons);
    const cookieHosts = this.hostsFromCookies(cookies);
```

```
  const localStorageHosts = this.hostsFromLocalStorage(localStorage);

  const hosts = this.inspectHosts(
    beaconHosts,
    cookieHosts,
    localStorageHosts,
    this.hostRecorder,
  );

  this.collectorOutput.websocketLog =
    this.webSocketRecorder.collectedWebSockets;
  this.collectorOutput.cookies = cookies;
  this.collectorOutput.localStorage = localStorage;
  this.collectorOutput.beacons = beacons;
  this.collectorOutput.hosts = hosts;

  return this.collectorOutput as EnrichedCollectionData;
}

private inspectCookies(): Array<InspectedCookie> {
  const { collectedCookies } = this.cookieRecorder;
  const collectorOutputCookies = this.collectorOutput.cookies;

  const cookiesFromEvents = flattenCookieEvents(collectedCookies);

  const mergedCookies = mergeCookieSources(
    cookiesFromEvents,
    collectorOutputCookies,
  );

  const markedCookies = markFirstPartyCookies(
    mergedCookies,
    this.firstPartyChecker,
  );

  if (this.cookieDb) {
    return sortCookiesByExpiration(
      categorizeCookies(markedCookies, this.cookieDb),
    );
  }
  return sortCookiesByExpiration(markedCookies);
}

private inspectLocalStorage(): InspectedLocalStorageSnapshot {
  const { localStorageEvents } = this.localStorageRecorder;
  this.collectorOutput.localStorage =
    this.localStorageRecorder.localStorageSnapshot;

  for (const origin of Object.keys(this.collectorOutput.localStorage)) {
```

```
    const parsedUrl = parseUrl(origin);
    const isFirstPartyStorage =
      parsedUrl && this.firstPartyChecker.isFirstParty(parsedUrl);

    const originStorage = this.collectorOutput.localStorage[origin];

    for (const key of Object.keys(originStorage)) {
      // add if entry is linked to first-party host
      originStorage[key].firstPartyStorage = isFirstPartyStorage;
      // find all logs for a given key
      const matchedEvents = localStorageEvents.filter(
        (event) =>
          origin == event.origin && Object.keys(event.data).includes(key),
      );

      if (matchedEvents.length > 0) {
        originStorage[key].logs ??= [];
        const { logs } = originStorage[key];

        for (const matchedEvent of matchedEvents) {
          if (
            !logs.some(
              (lsLog: EventMetaData & { timestamp?: string }) =>
                lsLog.location === matchedEvent.location,
            ) &&
            matchedEvent.location !== "about:blank"
          ) {
            originStorage[key].logs.push({
              location: matchedEvent.location,
              stack: matchedEvent.stack,
              timestamp: matchedEvent.timestamp,
              type: matchedEvent.type,
            });
          }
        }
      }
    }
  }

  return this.collectorOutput.localStorage;
}

private inspectBeacons(): Array<CollectedBeacon & { occurrences: number }> {
  const beacons = this.beaconRecorder.collectedBeacons;

  // make now a summary for the beacons (one of every hostname+pathname and their occurrance)
  const beacons_from_events_grouped: Record<
    string,
    Array<CollectedBeacon>
```

```
  > = groupBy(beacons, (beacon) => {
    const parsedUrl = parseUrl(beacon.url);
    return parsedUrl
      ? `${parsedUrl.hostname}${parsedUrl.pathname.replace(/\/$/, "")}`
      : "";
  });

  const beacons_summary = [];
  for (const [_, beacon_group] of Object.entries(
    beacons_from_events_grouped,
  )) {
    beacons_summary.push({
      ...beacon_group[0],
      occurrences: beacon_group.length,
    });
  }

  beacons_summary.sort((b1, b2) => b2.occurrences - b1.occurrences);

  return beacons_summary;
}

private inspectHosts(
  beaconHosts: Hosts,
  cookieHosts: Hosts,
  localStorageHosts: Hosts,
  hostRecorder: HostRecorder,
): {
  [key: string]: Hosts;
} {
  return {
    beacons: beaconHosts,
    contentSecurityPolicy: hostRecorder.contentSecurityPolicyHosts,
    cookies: cookieHosts,
    links: hostRecorder.linkHosts,
    localStorage: localStorageHosts,
    requests: hostRecorder.requestHosts,
  };
}

private hostsFromBeacons(
  beacons: Array<CollectedBeacon & { occurrences: number }>,
): Hosts {
  const hosts = {
    firstParty: new Set<string>(),
    thirdParty: new Set<string>(),
  };

  for (const beacon of beacons) {
```

```
    const beaconUrl = parseUrl(beacon.url);

    if (!beaconUrl || beacon.listName !== "easyprivacy.txt") continue;

    if (this.firstPartyChecker.isFirstParty(beaconUrl)) {
      hosts.firstParty.add(beaconUrl.hostname);
    } else {
      hosts.thirdParty.add(beaconUrl.hostname);
    }
  }
  return hosts;
}

private hostsFromCookies(cookies: Array<InspectedCookie>): Hosts {
  const hosts = {
    firstParty: new Set<string>(),
    thirdParty: new Set<string>(),
  };

  for (const cookie of cookies) {
    const parsedCookieUrl = parseUrl(
      `cookie://${cookie.domain}${cookie.path}`,
    );

    if (!parsedCookieUrl) continue;

    if (
      this.firstPartyChecker.isFirstParty(parsedCookieUrl) ||
      cookie.domain === ""
    ) {
      hosts.firstParty.add(cookie.domain);
    } else {
      hosts.thirdParty.add(cookie.domain);
    }
  }

  return hosts;
}

private hostsFromLocalStorage(
  localStorage: InspectedLocalStorageSnapshot,
): Hosts {
  const hosts = {
    firstParty: new Set<string>(),
    thirdParty: new Set<string>(),
  };

  Object.keys(localStorage).forEach((origin) => {
    const parsedUrl = parseUrl(origin);
```

```
      if (parsedUrl) {
        const isFirstPartyStorage =
          this.firstPartyChecker.isFirstParty(parsedUrl);

        const { hostname } = parsedUrl;

        if (isFirstPartyStorage) {
          hosts.firstParty.add(hostname);
        } else {
          hosts.thirdParty.add(hostname);
        }
      }
    });

    return hosts;
  }
}

export default Inspector;
```