

The aesthetic values of source code

Doctoral Symposium

Depaz Pierre

Universite Paris-3 Sorbonne-Nouvelle, Paris, France
pierre.depaz@sorbonne-nouvelle.fr

Abstract. This paper presents the state of my doctoral research on the aesthetics of source code. The approach taken here is that of literary and linguistic analysis: by treating source code as text, it becomes possible to identify discourses about what values such as clarity, elegance, simplicity, efficiency are essential to formal disposition of source code. Due to the ambivalent position of source code as language that is understandable both by humans and machines, I aim at highlighting the multiple relationships between those aforementioned values. Through this process of identifying how aesthetics manifests formally in source code, the role and influence of programming languages will also be touched upon, insofar as they provide the linguistic structure within which those aesthetic manifestations take place.

Keywords: Source code, aesthetics, programming languages, elegance, clarity.

1 Purpose of the research, its importance to the field and expected contributions

The purpose of this research is to identify the role that aesthetics play in the writing and reading of source code. At first somewhat counter-intuitive -why should something ultimately understood by a machine as binary voltage current be deemed "beautiful" or "ugly", "aesthetic" or not?- the fact that source code can have pleasing formal properties is almost universally acknowledged by programmers, digital artists and hobbyists alike. This implicit consensus about the possibility for code to be "beautiful" however falls short of providing identifiable and consistent formal aesthetic features specific to written source code. Taking the definition of aesthetics as a set of formal properties eliciting a sensory-emotional reaction in the creator and the beholder, this research project consists in looking at source code through the field of literary theory, by treating it as literary text, asking questions about form, authorship, readership, distribution and reception in the context of written scripts. The purpose of identifying the aesthetic properties of source code is to see to what extent these relate to, or differ from, the aesthetic properties of written language, and what is their relationship with the necessities of executing code. In particular, by looking at the tension between what is at the same time a machine-readable text and a human-readable text, I aim at highlighting the role that aesthetics play in both machine- and human-understandings of source code.

This research intends to open up new readings of source code, using the lens of aesthetics in order to better understand how code functions as a specific semantic apparatus. It intends to bypass approaches of code as a purely theoretical notion—"code" as a term encompassing any and all kinds of computation, and thus erasing the variety of programming languages that make up its reality—, or as an effective procedure—code as a compiled binary and executed software. This definition of source code as limited to written code thus excludes graphical programming languages, although it is possible that the conclusions of the current project could apply to those, as well as executed code, thus drawing influences from, but not directly contributing to, the field of electronic literature. Specifically, this research contributes to an approach of aesthetics as a functional component in the creation, understanding, and re-use of cognitive objects.

2 Brief survey of background and related work

Overall, I intend to draw from two separate fields of research which address aesthetics in programming. The first

starts with the literature on software development and software engineering as technical practice, as illustrated by the works of Dijkstra, Knuth, Kernighan and others. Most of the publications in this field connects aesthetics, as well as the related fields of "art of computer programming" (Knuth, 1962), "craft of programming" (McBreen, 2001), or "beautiful code" (Oram, 2007), to the reality of writing and reading understandable code—in a sense being concerned with "applied aesthetics". If one of the greatest hurdles of programming is to be able to comprehend what a piece of software's potential is, aesthetics is approached here as an essential step towards overcoming that hurdle.

A second field stems from media studies and software studies. It starts from broad theoretical work establishing a definition of software as a socio-technical and cultural object (MacKenzie, 2006), towards close, deliberate readings of source code in more recent work (Montfort et. al., 2012, Paloque-Berges, 2009), complemented by an approach from the field of literary studies (Marino, 2020, Hayles, 2005). The object of source code is more and more the focus of researchers, particularly in terms of *what code means*, how it represents the world, and how it is represented in the world. However, little attention has been given to what kind of intrinsic formal aesthetic properties it could exhibit, or to what standards can source code be held to as an aesthetic object.

While dealing with the same object, the corpora studied are clearly separated from one another. On the one side, there exists a well-established corpus of commented source code circulating in the professional field of software development. These include best practices, code demonstrations, commentaries and text books, ranging from Dijkstra's *programming pearls* (Dijkstra, 1970) to the commentary of the Unix version 6 source code (Lions, 1976). The intent of this research is to further improve the quality of the software made by industry practitioners. References to other fields (such as linguistics, literary studies, or cognitive psychology) are therefore rare but not completely absent (see: Knuth, 1984, Weinberg, 1998). Most of the contributions of those texts is to establish concrete best practices when writing and reading source code, yet they only accidentally link it back to aesthetic and literary concepts such as *simplicity*, *clarity*, and *elegance*.

On the other side, the body of texts of self-defined *source code poetry* has been given close attention, but yet again mostly in terms of socio-cultural practices, and what they mean within the communities of programmers within which they were written and read. While often addressing the *poetics* of code (i.e. the world-making, storytelling potential of software), some of these (see: Temkin, 2007, Aquilina, 2015) focus particularly on formal exercises such as Perl Poetry, the Obfuscated C Code Contest and well-known esoteric languages and investigate to what extent these writings could qualify as "code". Within the realm of "artistic code", Paloque-Berges's work is closest when it comes to analyzing the specific forms that source code poetry takes, and what kinds of relationships those forms have with natural languages. For instance, she does so both by highlighting syntactical tokens that make Perl uniquely suited to this kind of endeavour, but also by providing larger theoretical frameworks, based on the idea of fiction (Goodman, 1978) and of practice (De Certeau, 1990).

The most relevant and interesting research today has been coming from the field of *Critical Code Studies* (Marino, 2020) and digital rhetoric. These link directly the syntactic and the semantic components of software: reading source code is essential to understanding what that code is intended to do. It is in this trajectory of code rhetoric that I place this research project, particularly in terms of untangling the relationship between specific formal manifestations and overall conceptual understanding. In that sense, I will be investigating this specific overlap by linking both the analysis of software as a technical practice and as a socio-cultural object.

3 Description of the proposed approach and current progress

The proposed methodology is threefold: empirical, theoretical and experimental. The basis of this research will be constituted of primary sources: source code available online, published in hobbyist magazines, commented in textbooks or published as artworks. The analysis of these texts, along with the discourses around those texts (comments, reviews, discussions) will help develop a heuristic to understand when and how the criteria of the beautiful, as well as the ugly, is summoned among communities of programmers. This corpus encompasses all ranges of project sizes, languages and periods, from the Apollo IX landing module procedure, to the snippets published in fanzines, or the examples shown in style guides, as well as the more obvious recent publications of

source code poetry. The wide variety of texts integrated into the corpus will allow me to identify to what extent the aesthetic standards identified are context-free—i.e. which aspects vary between high-level languages and low-level languages? between professional and amateur codebases? between educational and artistic settings?

The second component of this process will be to examine the findings from the analysis of this corpus at the light of the somewhat more traditional concepts of literary theory and linguistics. Recurring concepts in programming discourses, such as *clarity*, will be further examined in the light of concepts such as authorship, reception, rhetorical figures, style, voice and layout, among others. This cross-disciplinary approach will establish a framework which takes into account both the appreciations intrinsic to the communities of practice of source code (e.g. software developers), as well as further highlight some unique aesthetic properties of code via literary methods of analysis. For instance, there could be an exploration of the concept of *simplicity* by comparing the programming paradigm *DRY* (Don't Repeat Yourself) and Barthes's *writing degree zero* (Barthes, 1972), or a re-assessment of *explicitness* in the light of Mikhail Bakhtin's *dialogism* (Bakhtin, 1981).

Having sketched a conceptual framework for approaching source code as a text with potentially aesthetic manifestations, the last part of this research project will consist in confronting these conclusions with case studies. In line with the empirically starting point and the close-reading practice of Critical Code Studies, I intend to examine three texts, ranging in variety in terms of source code quality (professional, amateur, artistic), in order to confirm or infirm, and in any case further elaborate on, the framework established in the previous stage.

By working across disciplines, this research project intends to contribute both to software studies and to aesthetics. In terms of software studies, I have compiled a formal account of the collective practices and judgments which constitute aesthetic standards in the practice of reading and writing programs, highlighting to what extent these standards are contextual, and to what extent they are general, and in general making explicit the "ambient knowledge" which exists within these communities of practice. Following De Certeau's work on cultural studies and Sennett's work on craftsmanship, the next step is to show that the act of writing and reading beautiful code *matters*.

So far, showing that aesthetics matters has led me to further qualify the role of aesthetics as a functional one (telic, rather than autotelic), insofar as aesthetics can be judged by the extent to which, by making something *understandable*, they make something *work*. This research project therefore aligns with Goodman's claim that aesthetics are a functional part of cognitive appreciation of objects and phenomena with which we engage. By focusing on the relationships between the made (the object), the said (the language, programming or natural) and the thought (the concept) in the unique combination that software offers, I expect to contribute to a deeper understanding of the role of human and machine linguistics, as well as social contexts (as they are presupposed by linguistic activity) in the appreciation and judgment of aesthetic forms.

References

- Aquilina, Mario**, 2015. *The Computational Sublime in Nick Montfort's 'Round' and 'All the Names of God'*, in *CounterText*, Vol. 1 No. 3, Edinburgh University Press.
- Bakhtin, Mikhail**, 1981. *The Dialogic Imagination: Four Essays*, Austin and London: University of Texas Press.
- Barthes, Roland**, 1972, *Le Degré zéro de l'écriture suivi de Nouveaux essais critiques*, Paris: Editions du Seuil.
- Chandra, Vikram**, 2014, *Geek Sublime: The Beauty Of Code*, Port Townsend, WA: Graywolf Press.
- Cramer, Florian**, 2005, *Words Made Flesh: Code, Culture, Imagination*, Rotterdam: Piet Zwart Institute.
- De Certeau, Michel**, 1990, *L'Invention du Quotidien*, Paris: Editions Gallimard.
- Dijkstra, Edsger**, 1970. *Notes On Structured Programming*, TU Eindhoven, retrieved online from: <https://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF> on 28/04/2020.
- Goodman, Nelson**, 1978, *Ways of World-making*, Indianapolis, IN: Hackett Publishing Company.
- Hayles, N. Katherine**, 2005. *My Mother Was a Computer: Digital Subjects and Literary Texts*, IL: University of Chicago Press
- Knuth, Donald E.**, 1962, *The Art of Computer Programming*, MA: Addison-Wesley.
- Lions, John**, 1976. *Lions' Commentary on UNIX 6th Edition, with Source Code*, University of New South Wales.
- Mackenzie, Adrian**. 2006. *Cutting Code: Software and Sociality*. New York, NY: Peter Lang.
- Marino, Mark**, 2020. *Critical Code Studies*, MA: MIT Press.
- McBreen, Peter**, *Software Craftsmanship: The New Imperative*, MA: Addison-Wesley.
- Montfort, Nick et. al.**, 2012, *10 PRINT CHR\$(205.5+RND(1)); : GOTO 10*, Boston, MA: The MIT Press.
- Oram, Andy, Wilson, Greg**, 2007, *Beautiful Code: Leadings Programmers Explain How They Think*, BE: O'Reilly.
- Paloque-Berges, Camille**, 2009. *Poétique des codes sur le réseau informatique*, Paris, Editions des Archives Contemporaines.
- Temkin, Daniel**, 2017. *Language Without Code: Intentionally Unusable, Uncomputable, or Conceptual Programming Languages*, in *CITAR*, Vol. 9, No. 3, Portuguese Catholic University.
- Weinberg, Gerald**, 1998. *The Psychology of Computer Programming*, NY: Dorset House.