The role of Aesthetics in the Understandings of Source code

Pierre Depaz
under the direction of Alexandre Gefen (Paris-3)
and Nick Montfort (MIT)

ED120 - THALIM

last updated - 04.10.2021

1 Introduction

This thesis is an inquiry into the formal manifestations of source code and how particular configurations of lines of code allow for aesthetic judgments. The implications of this inquiry will lead us to consider the different ways in which people who read and write code, and through these, we will explore the different ways in which source code can be represented, depending on what it aims at communicating. This study on source code involves the different groups of people which read and write it, the purposes for which they write it, the languages they use to write it, and the language they use to speak about it. Most importantly, this thesis focuses on source code as a material, linguistic manifestation of a larger digital ecosystem of software and hardware to which it belongs. Since source code isn't code, as we will see below, this thesis also aims at studying the reality of written code, rather than its conceptual interpretations.

Starting from pieces of source code, henceforth called *program texts*[1], this thesis will aim at assessing what programmers have to say about it, and attempt to identify one or more specific *aesthetic registers*. This aim depends on two facts: first, source code is a medium for expression, both to express the programmer's intent to the computer[2] and the programmer's intent to another programmer[3]. Second, source code is a relatively new medium of expression, compared to either fine arts or engineering practices. As a recent medium for expression, the development and solidification of aesthetic practices—that is, of ways of doing which do not find their immediate justification in a practical accomplishment—is an ongoing research project in computer science, software development and more recent fields within the digital humanities. Formal judgments of source code are therefore existing and well-documented, and they are related to a need for expressiveness, as we will see in chapter X, but their formalization is still an ongoing process.

Source code thus has ways of being presented which are subject to

aesthetic judgments by programmers; that is, code *has* aesthetics, but it is unclear exactly *which* aesthetics. Indeed, these aesthetic judgments as they exist today rely on different domains to assess source code, as a means to grasp the cognitive object that is software. These draw from metaphors which range from literature, architecture, mathematics and engineering. And yet, source code, while qualified on all of these, source code isn't specifically any of these. Liked the story of the seven blind men and the elephant[4], each of these domains touch on some specific aspect of the nature of code, but none of them are enough to entirely provide a solid basis for the aesthetic judgments of source code. It is at the crossroads of these domains that this thesis locates its work.

The examination of source code, and of the discourses around source code will integrate both the myriad of ways in which source code can exist, and the invariant aspects which underline all diverse approaches of source code. Particularly, we will see how each groups of practicionners tend to deploy references to one particular set of metaphorical references drawing from the domains above, but also how these references overlap across groups. The point of overlap, as we will demontrate, is that of *using a formal linguistic system to communicate the understanding of complex cognitive structures*. Relying heavily on Nelson Goodman's work on the languages of art[5], we end on connecting this to the broader role of aesthetics as a cognitive mechanism to deal with complexity.

The rest of this introduction will consist in establishing a more complete view of the context in which this research takes place, from computer science to digital humanities and science and technology studies. With this context at hand, we will proceed to highlight the specific problems which will be tackled—the current place of aesthetics in source code. After outlining our methodology and the theoretical frameworks which will be mobilized throughout this study, we will sketch out how the different chapters of this thesis will attempt at providing some responses to our research questions.

1.1 Context - 15p

1.1.1 The research territory: code

Most of our modern infrastructure depends, to a more or less dramatic extent, on computer systems[6], from commercial spaces to classrooms, transport systems to cultural institutions, scientific production and entertainment products. The complex processes are described in what is called source code, and the number of lines of code involved in running these processes is hard to estimate; one can only rely on disclosures from companies, and publicly available repositories. For instance, all of Google's services amounted to over two billions source lines of code (SLOC)[7], while the 2005 release of the OSX operating system comprised 86 millions lines of code, and while the version 1.0 of the Linux kernel (an operating system which powers most of the internet and specialized computation) totalled over 175,000 SLOC, version 4.1 jumped to over 19.5 million lines of code in the span of twenty years[8].

Who reads this code? To answer this question, we must start diving a little bit deeper into what source code really is.

At a high-level, source code consists in a series of instructions, composed in a particular programming language, which is then processed by a computer in order to be executed. For instance, using the language called Python, the source code:

```
a = 4
b = 6

def compute(first, second):
    return (first * 2) + second

compute(a, b)
```

consists in telling the computer to store two numbers in what are called *variables*, then proceeds with describing the *procedure* for adding the dou-

ble of the first terms to the second term, and concludes in actually executing the above procedure. Given this particular piece of source code, the computer will output the number `14` as the result of the operation `(4 * 2)+ 6`. In this sense, then, source code is the requirement for software to exist. If computers are procedural machines, acting upon themselves and upon the world, they need a specification of what to do, and this specification exists in the form of source code.

Source code is here both a requirement and a by-product, since it isn't required anymore once the computer has processed and stored it into a *binary* representation, a series of 0s and 1s which represent the successive states that the computer has to go through in order to perform the action that was described in the source code. *Binary code* is what most of the individuals who interact with computers deal with, and (almost) never have to inquire about, or read its source code. On one hand, then, source code only matters until it gets processed by a computer, through which it realizes its intended function.

On the other hand, source code isn't just about telling computers what to do, but also about a particular economy: that of software development. Software developers are the ones who write the source code and this process is first and foremost a collaborative endeavour. Software developers write code in successive steps, because they add features over time, or they fix errors that have shown up in their software, or they decide to rewrite parts of the source code based on new ideas, skills or preferences. In this case, source code is not used to communicate to the computer what it does, but it is used to communicated to other software developers what the *intent* of the software is. Source code is then the locus of human, collaborative work; it represents iterations of ideas, formalization of processes and approaches to problem-solving.

Official definitions of source code straddle the line between the first role of source code (as instructions to a computer) and the second role of source code (as indications to a programmer). For instance, a definition

within the context of the Institute of Electrical and Electronics Engineering (IEEE) is that of *any fully executable description of a software system, which therefore includes various representations of this description, from machine code to high-level languages and graphical representations using visual programming languages*[9]. This definition focuses on the ability of code to be processed by a machine, and mentions little about its readability (i.e. processability by other humans).

On the other hand, the definition of source code provided by the Linux Information Project[1] focuses on source code as *the version of software as it is originally written (i.e. typed into a computer), by a human in plain text (i.e. human-readable, alphanumeric characters).*[10]. The emphasis here is on source code as the support of human activity, as software developers need to understand the pieces of code that they are creating, or modifying. Source code thus has two kinds of readabilities: a computer one, which is geared towards the correct execution of the program, and a human one, which is geared towards the correct understanding of the program. In the lineage of this human-readability, we can point to the Free Software Foundation's equation of the free circulation and publication of source code with the free circulation of publication of ideas. Particularly, Freedom 1 (*The freedom to study how the program works, and adapt it to your needs*) and Freedom 2 (*The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.*[11]) as stated in the FSF's definition of Free Software stipulates that access to source code is required to support these freedoms, a version of source code that is *not concealed*, i.e. readable by both human and machine.

In addition to this ability for source code to communicate the ideas latent in it, source code, as an always potentially collaborative object, can be the locus multiple subjectivities coming together. As Krysa and Sedek state it

[1] https://linfo.org/sourcecode.html

in their definition, *source code is where change and influence can happen*, and where intentionality and style are expressed[12]. In their understanding, source code shares some features with natural languages, and as such is different from the machine language representation of a program. Its intelligibility, they continue, then facilitates its circulation and duplication among programmers.

In this research, we build on these definitions to propose the following:

> Source code is defined as one or more text files which are written by a human or by a machine in such a way that they elicit a meaningful response from a digital compiler or interpreter, and describe a software system. These text files are the starting point to produce an execution of the system described, whether the very first starting point, or an intermediate representation (used for subsequent compilations).

This definition takes into account a broad view of source code, including steps such as intermediate representations (transitoriy representations from one version of the source to another one), but also obfuscations (deliberately complicating the code to prevent human-readability while maintaining machine-readability) and minifications (reducing the amount of characters used in source code to its minimum). This will allow us to compare human-authorship of source code, machine-authorship, and hybrid modes, in which a human writes unreadable code with the help of tools. One aspect that is being excluded from this definition is the actual manifestation of code: while multiple media for source code exist, we exclude here all of those that are not written in the UTF-8 character set. Since one of the questions of this study is to examine the literariness of source code aesthetics, other forms of source code, such as visual programming languages or biological computation, stand outside the scope of this study and should be investigated in subsequent work.

1.1.2 Beautiful code

Under this definition of source code textually represented, we now turn to the existence of the aesthetics of such *program texts*. To frame this existence, we first need to touch upon the history and practice of software development. As an economic activity, software development came from a bottom-up dynamic, a *de facto* activity which was not expected in the early days of computing. Its earliest manifestation can be found in the physical rewiring process of mainframes in order to perform a specific computation, something more akin to firmware than to software. These rewiring tasks were done by mostly female assistants, under the direction of mostly male mathematicians[13], and considered a simple translation task which did not need any particular attention. The recognition of software engineering as its own field came as its unique domain of expertise was required in larger engineering projects—the term *software engineering* was coined in the late 1960s by Margaret Hamilton and her team as they were working on the Apollo 11 Lunar Module software[14]. In the same decade, the first volume of *The Art of Computer Programming*, by Donald Knuth, addresses directly both the existence of programming as an activity separate from both mathematics and engineering, as well as the ”artistic” component of such an activity[15]. The volume opens on the following paragraph:

> The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music. This book is the first volume of a multi-volume set of books that has been designed to train the reader in the various skills that go into a programmer's craft.[15]

Considered one of the most canonical textbooks in the field, *The Art of Computer Programming* lays out two important aspects of programming:

that it can be an aesthetic experience and that it is the result of a craft, rather than of a highly-formalized sytematic process.

Craftsmanship as such is an essentially fleeting phenomenon, a practice rather than a theory, in the vein of Michel De Certeau's *tactics*, bottom-up actions designed and implemented by the users of a situation, product or technology as opposed to *strategies*[16], in which ways of doing are prescribed in a top-down fashion. It is hard to formalize, and the development of expertise in the field happens through practice as much as through formal education[17]. The domain of craft is also one in which function and beauty exist in an intricate relationship, based on subjective qualitative standards rather than strictly external measurements, even thought the former are rarely explicited[18].

Approaching programming (the activity of writing and reading code) as a craft[19] connects to the multiple testimonies of encountering beautiful code, some of which have made their ways into edited volumes or monographs[20, 21, 22]. Additionally, informal exchanges among programmers on forums, mailing lists, blog posts and code repositories often mention beautiful code, either as a central discussion point or simply in passing. It is these testimonies, from textbooks to online posts, which constitute the first part of our corpus, as documents of programmers commenting on their practice of reading and writing code. The second part of the corpus is consituted of selected program texts, source code text files which we will examine closely in order to formalize which aspects of the textual manifestation of software elicits an aesthetic experience in those who read and write it.

- ~~start with a very basic description of what code/source code/software is~~
- ~~quick history of software development~~
- ~~demonstrate that program texts are a thing then that programmers know there is beautiful code.~~

- connect the ”textual manifestation” aspect with my definition of aesthetics.
- then about what computer science, programming, humanities, etc.

1.1.3 The definitions

- Aesthetics
- ~~Source Code~~
- ~~Program Text~~
- Understanding

1.1.4 Boundaries

that which i shall not touch

but once we have laid out all of this, then it becomes possible to highglight the gaps:

- the assumption of the literarity of code
- the lack of theoretical framework explaining why some code is beautiful and why other isn’t
- the lack of *proof*, which is showing code
- the apparent assumption that aesthetics have nothing to do with source code

1.2 Problem - 10p

start by establishing, based on what was said previously, the *niche* that i will occupy

- the fetishization of code

- the multiplicity of registers
- the reason why it matters (connecting beautiful and useful)

then really establish the significance of this study: beauty and function

1.3 Methodology - 10p

- reading code
- reading discourses
- reading aesthetic theory based from these discourses
- also the theoretical frameworks

1.4 Roadmap - 6p

list all chapters with a brief overview for each.

1.5 Connecting back to the wider world - 2p

and readership

References

[1] Francoise Detienne. *Software Design – Cognitive Aspect*. Springer Science & Business Media, December 2012.

[2] Edsger W. Dijkstra. Chapter I: Notes on structured programming. In *Structured programming*, pages 1–82. Academic Press Ltd., 1972.

[3] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs - 2nd Edition*. Justin Kelly, 1979.

[4] Wendy Hui Kyong Chun. On "Sourcery," or Code as Fetish. *Configurations*, 16(3):299–324, 2008. Publisher: Johns Hopkins University Press.

[5] Nelson Goodman. *Languages of Art*. Hackett Publishing Company, Inc., Indianapolis, Ind., 2nd edition edition, June 1976.

[6] Rob Kitchin and Martin Dodge. *Code/Space: Software and Everyday Life*. The MIT Press, 2011.

[7] @Scale. Why Google Stores Billions of Lines of Code in a Single Repository, September 2015.

[8] Linux kernel, October 2021. Page Version ID: 1048584104.

[9] Mark Harman. Why Source Code Analysis and Manipulation Will Always be Important. In *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*, pages 7–19, September 2010.

[10] Source code definition by The Linux Information Project.

[11] Richard Stallman and Mass) Free Software Foundation (Cambridge. *Free software, free society : selected essays of Richard M. Stallman*. Boston, MA : Free Software Foundation, 2002.

[12] Matthew Fuller, editor. *Software Studies: A Lexicon*. The MIT Press, Cambridge, Mass, April 2008.

[13] Wendy Hui Kyong Chun. On Software, or the Persistence of Visual Knowledge. *Grey Room*, 18:26–51, January 2005.

[14] David A. Mindell. *Digital Apollo: Human and Machine in Spaceflight*. MIT Press, September 2011. Google-Books-ID: gXYItzQARVoC.

[15] Donald E. Knuth. *The Art of Computer Programming, Volume 1 (3rd Ed.): Fundamental Algorithms*. Addison Wesley Longman Publishing Co., Inc., USA, 1997.

[16] Michel de Certeau, Luce Giard, and Pierre Mayol. *L'invention du quotidien*. Gallimard, 1990.

[17] Richard Sennett. *The Craftsman*. Yale University Press, 2009.

[18] David Pye. *The Nature and Art of Workmanship*. Herbert Press, illustrated edition edition, July 2008.

[19] Pierre Lévy. *De la programmation considérée comme un des beaux-arts*. Textes à l'appui. Anthropologie des sciences et des techniques. Éd. la Découverte, Paris, 1992.

[20] Andy Oram and Greg Wilson, editors. *Beautiful Code: Leading Programmers Explain How They Think*. O'Reilly Media, Beijing ; Sebastapol, Calif, 1st edition edition, July 2007.

[21] Vikram Chandra. *Geek Sublime: The Beauty of Code, the Code of Beauty*. Graywolf Press, September 2014.

[22] Richard P. Gabriel. *Patterns of Software: Tales from the Software Community*. Oxford University Press, 1998.