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 result in 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. Since source code isn't code, 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 aims at assessing what other programmers have to say about it, and attempt to identify a specific *aesthetic register*. 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 the 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; more specifically, code *has* aesthetics, but it is unclear *which* aesthetics. Indeed, these aesthetic judgments as

they exist today rely on different domains to apprehend what source code is as a means to apprehend the cognitive object that is software, drawing from metaphors which range from literature, architecture, mathematics and engineering. And yet, source code, while drawing on all of these, source code isn't specifically any of these. Liked the story of the seven blind men and the elephant[4], it is likely that 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 starts its work.

The examination of source code, and of the discourses around source code will reveal 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, structured systems*. 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 everyday life. With this context at hand, we will proceed to highlight the specific problems which will be tackled—that is, the 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 number of lines of code involved in running these complex 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) 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, the source code:

```
a = 4
b = 6

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

computer(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 double 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.

In this sense, source code is 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 suite 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 bother about 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. First of all, software developers write code in successive steps, because add features over time, or fix errors that have shown up in their software, or 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 defintions 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 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.

- Talk about readability more with the use of FSF's definition
- Then mention Software studies' definition, which involves style and modification and intent
- then conclude with my definition of source code, by highlighting the original aspect and the textual aspect

I define source code 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).

- ~~start with a very basic description of what code/source code/software is~~
- then that programmers know there is beautiful code.
- then about what computer science, programming, humanities, etc.

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

1.1.2 The definitions

- Aesthetics
- Source code
- Program Text
- Understanding

1.1.3 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

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.