Beauty is Truth, Truth Beauty

Aspects of Philosophy and Aesthetics in Computer Programming

Vesa Norilo
University of Arts Helsinki
March 29, 2016

Abstract

This essay discusses some aspects of philosophy and aesthetics applied to computer science. In particular, what makes programs ugly or beautiful, and how that relates to the philosophical traditions of aesthetics.

Keywords: aesthetics, philosophy, computer science, programming

Introduction

Computing devices have become ubiquitous to the degree that the computation has become invisible – the user of mobile phones, computers and televisions rarely has to be aware of the computational processes that underpin these machines. While automated data processing and computation is sometimes taken as a feature of the modern times, the idea of computing machines is ancient, predating mathematical written notation.

Ancient Computers

Current understanding is that the *abacus* was the first widely estabilished computational device (Ifrah, 2001), appearing as early as 2700–2300 BC in Sumer. It was known to most ancient civilizations. The counting tokens of the Roman abacus, *calculi*, became the very symbol and namesake for calculation.

The use of mathematics in the ancient world had both mundane and transcendental applications. On of the most sophisticated early mechanical calculators was the Antikythera mechanism, which was likely use for

astronomical purposes (Freeth, 2006). On the other hand, early equational reasoning was codified in the 9th century in Persia by Muḥammad ibn Mūsā al-Khwārizmī (محمد بن موسى الخوارزمى) with the primary application in economics: the share and transfer of land ownership.

Analytical Engine

With applications such as navigation placing increasing emphasis on trigonometry, more sophisticated computational aids became necessary. To facilitate trigonometric calculations in reasonable time, mathematical tables and interpolation was used. However, construction of mathematical tables by hand was error-prone.

This prompted Charles Babbage (1791–1871) to devise an automatic calculator called the Difference Engine for production of mathematical tables. Babbage's plans to produce the Difference Engine, as well as its successor, the *Analytical Engine*, a generalized mechanical computer, were thwarted by the lack of funding and the lack of precision in the mechanical production of his time (Halacy, 1970).

Of Objects and Meta-Objects

Babbage's 19th century development effort faced similar practical considerations as the manufacture of integrated circuits did 150 years later. The financial implications of manufacturing a failed design forced Babbage to develop a *mechanical notation* – a formal notation of flow diagrams describing mechanical computers. This is akin to contemporary hardware description languages used in the design of integrated circuits. As of this writing, the correctness of Babbage's notation for the Analytical Engine is not verified (Johnstone, 2014).

Johnstone (2014) argues that the mechanical notation is even more remarkable than the physical difference engine itself. He describes Babbage as the "first individual to work with systems where the function transcends the components" – a system in which the meta-object of the mind looms larger than the physical, tangible object. The complex state-space of a apparatus like the difference engine renders its physical manifestation or even the construction plans less than useful in understanding its fundamental nature and purpose. In contrast, the mechanical notation *is* the meta-object – essentially, a data flow graph of pulleys, levers and switches. The mechanical notation can be used to produce plans and eventually a physical manifestation of the meta-object.

Universal Meta-Computers

Babbage's ambition for the Analytical Engine was to produce a *general* automated calculator. While the Difference Engine had a clear industrial and military purpose, the Analytical Engine represents a desire for an all-knowing oracle, an augment for human intelligence, an unerring instrument for the perpetual quest for knowledge.

Such an omnipotent meta-object was proposed by Alan Turing in 1936 (Hodges, 2012). The *Turing Machine* is a mathematical representation of a simple device that is nevertheless capable of simulating an arbitrary computer algorithm. Turing devised the machine when working on the *halting problem* first presented by David Hilbert in the context of Diophantine equations and later understood more generally – Turing proved the problem unsolvable.

Robin Gandy (1919-1995) – a student of Turing's – argued that Charles Babbage's *Analytical Engine* already contained the principle of a Turing machine, with a near-orthogonal basis of operators. The important distinction is that Babbage's mechanical notation deals with concrete physical objects – the building blocks of his mechanical computers. Turing machine, on the other hand, is primarily a *meta-object*. Even though it carries over metaphors from the tangible world of Turing's experience, like storage tape and mechanical motors, its intention is not to be a plan for an actual computer, but to *model computation itself*.

In the 1930s, Alonzo Church (1903-1995) proposed λ-calculus, a formal system in mathematical logic based on function abstraction and application. In 1936 Church proved, independently of Turing, that Hilbert's *halting problem* was unsolvable. In 1952, Stephan Kleene, a student of Church's, would demonstrate that Church's and Turing's computability models were in fact isomorphic, and formulate them as the *Church-Turing thesis* (Kleene, 1981). Remarkably, the Turing machine and λ-calculus are equivalently powerful models of computation. Church's calculus discards the connection to the tangible world, and has not even a tenuous resemblance to a physical entity.

Mechanisms and Magicks

The Turing machine and λ-calculus represent, despite each being able to encode the other, very different attitudes to the fundamentals of computation. There is an analogue to the traditions of science described by Kearney (1971): the Turing machine is a prototypical mechanist device, the *how* of computation, a meticulous exhibit of the minimalist aesthetic of engineering. λ-calculus, on the other hand, has features of the magical: function com-

position and constructs of incomprehensible depth, outside time and space: the *what* of computation.

FORTRAN and LISP

Some of the early successful programming languages can be seen as aspects of the mechanist and magical strains of computer science. In 1954, John W. Backus designed FORTRAN – acknowledged as the first high-level programming language (Backus, 1979). FORTRAN can be seen as a direct extension of a Turing machine: a chronological sequence of operations that manipulate the state of a memory device. FORTRAN was designed to improve the productivity of programmers working on the various cumbersome machine language dialects of the time.

Soon afterwards, the programming language LISP was devised by John McCarthy in 1958 (Steele, 1984). It is modeled after the λ-calculus, and based entirely on function abstraction and application. Remarkably, McCarthy did not implement LISP or even believe that it could be done. The conception of LISP eval as an universal function, by Steve Russell in 1958, is often described in a way that evokes a revelation – a sudden manifestation of an implementation due to a breakthrough insight.

Aesthetics of Programming

A programmer's choice of a language is influenced by several reasons. These often arise from viewing programming as a primarily economic, commercial and industrial activity. If a computer program is considered as a form of human expression, different aspects of programming move to the forefront.

Kleene's isomorphism Kleene (1981) proves that semantically similar programs can be written in any programming language that is translatable to either the Turing machine code or λ-calculus. This includes the vast majority of past and present programming languages. If the functionality of a computer program is considered to be its primary feature, the choice of programming language is largely inconsequential. However, most programmers feel very strongly about their choice of tools, languages and idioms. This suggests that programs, as a form of human expression, have more to them than their outward function. In the subsequent sections, I propose aesthetics as a possible factor in the choice and valuation of programming idioms.

British aesthetics in the 18th century

According to Shelley (2014), the major works on aesthetics in the 18th century Britain can be categorized in three groups: internal-sense theories, imagination theories and association theories. Shaftesbury, the influential proponent of internal-sense, posits that proper appreciation of beauty should be *rational* and that it should seek the *original*, rather than the representation. In the context of computer programs, Shaftesbury could be understood to say that any beauty in a program is the beauty of the mind that devised it – perhaps compounded by the mind that designed the programming language. However, this would appear to be *representative* beauty, since the program is not the mind, and even the mind was taken by Shaftesbury to merely represent the beauty of its divine creator.

Hutcheson was greatly influened by Shaftesbury, and built on his theory by reinforcing the concept of an internal beauty-sense. Whereas Shaftesbury argued that beauty is a trait of a mind, recognized by a mind, Hutcheson claims that the "Power of Perception" that recognizes beauty is a sense like the other senses – and that beauty is not only of the mind or of the representation of it (Shelley, 2014). Both Shaftesbury and Hutcheson maintain that things are beautiful due to their proportion or order, "uniformity amidst variety", which certainly could be considered in the context of computer programs.

Reid articulated Hutcheson's idea about sensing beauty thus (in Shelley 2014):

> Beauty or deformity in an object, results from its nature or structure. To perceive the beauty therefore, we must perceive the nature or structure from which it results. In this the internal sense differs from the external. Our external senses may discover qualities which do not depend upon any antecedent perception …. But it is impossible to perceive the beauty of an object, without perceiving the object, or at least conceiving it.

Applying Reid, the beauty of a programming language would result from its nature and *structure*, something which must be perceived and conceived by the programmer in order for him or her to appreciate them.

In contrast to the internal-sense theorists, Addison posits that the experience of beauty stems from imagination. Burke built on the imagination theory by introducing an aesthetic dualism: that of the beautiful, and that of the sublime. According to Burke, beautiful excites the societal passion of love, sublime excites the self-preservative passion of astonishment. Beautiful things tend to be small, smooth, various, delicate, clear and bright: sublime things are great, uniform, powerful, obscure and somber (Shelley, 2014).

German aesthetics in the 18th century

Wolff was an early philosopher in the German tradition of aesthetics. He was inspired by the mathematics and philosophy of Leibniz. In addition to constructing a systematic philosophy out of Leibnizian thinking, he also contributed significant original ideas. Wolff's definition of beauty as perfection has strong parallels to computer programming. Perfection is formally defined as the harmony and ordering of parts in a whole, but in addition, substantively, as the suitability of the ordering in achieving that which was the aim for the whole (Guyer, 2014). Wolff's beauty has utilitarian aspects: the appeal of the refined body of an athlete, the intricate working of a mechanical device, or by analogy, the harmonious construction of a well-performing computer program. A beautiful building is a "space that is enclosed by art in order that certain functions can proceed there securely and unhindered" (in Guyer 2014).

Wolff defines distinctness and indistinctness in cognition: thoughts are clear when they are well known and distinguishable from other things. They are obscure when indistinct (Guyer, 2014). This resonates with some best practices in acknowledges in commercial programming, such as separation of concerns and modularity.

Gottsched proposes beauty as the "sensitive cognition of perfection". Judgements of taste are not due to "wit, imagination or memory", nor to any "sixth sense" – "understanding" but not "reason". According to Gottsched, the judgement of taste tracks the rules that the experts intuitively know, but are not governed by those rules (Guyer, 2014). Gottsched thus comes close to defining what is considered tacit or embodied knowledge in the current literature (Doy, 2008). Programming can be considered as a craft, thus exemplifying the accumulation of tacit and embodied knowledge that cannot be easily verbalized. Some of the vocabulary used by craftsman programmers reflects this: "code smell" is a term widely used to describe design that is certainly wrong, but hard to attribute to any specific detail, pattern or technique that was used. The choice of the word "smell" can be considered as a strong aesthetic judgement.

Baumgartner, who coined the term "aesthetics" in 1735, defines aesthetics as the analogue of rational cognition (in Guyer 2014):

> I cognize the interconnection of some things distinctly, and of others indistinctly, consequently I have the faculty for both. Consequently I have an understanding, for insight into the connections of things, that is, reason (ratio); and a faculty for indistinct insight into the connections of things, which consists of the following: 1) the sensible faculty for insight into the concordances among things, thus sensible wit; 2) the sensible faculty

> for cognizing the differences among things, thus sensible acumen; 3) sensible memory; 4) the faculty of invention; 5) the faculty of sensible judgment and taste together with the judgment of the senses; 6) the expectation of similar cases; and 7) the faculty of sensible designation. All of these lower faculties of cognition, in so far as they represent the connections among things, and in this respect are similar to reason, comprise that which is similar to reason (analogon rationis), or the sum of all the cognitive faculties that represent the connections among things indistinctly.

Aesthetic judgements are separate from intellect, but similar as functions of cognition – sensible cognition.

Structure, symmetry and order

A recurring theme in early aesthetics is *perfection* with a divine connotation. Various writers describe perfection as symmetry, harmony and order. While the λ-calculus and the Turing machine are celebrated for their universality – capability of expressing any computation – the progress of computer science since then has focused on what *not* to express. Structured programming (Clark et al., 2000, p. 20) is a paradigm introduced in ALGOL in the late 1950s explicitly to rule out certain methods of flow control, favoring logically structured nested program blocks instead of the freeform flow of the Turing tape.

This imposes a particular symmetry on programs, which could perhaps be even called harmony. Structured programming, like many inventions in computer science, is typically described in economic terms: reduction of errors, improving productivity. However, the parallel to the aesthetics of order and symmetry is evident, so much so that the use of non-structured methods, such as the *goto* statement, are almost universally frowned upon (Dijkstra, 1968), even in contexts in which there is no rational basis for such a preference.

Dewey: Experience and Expression

According to Leddy (2016) John Dewey, an American pragmatist, is best known from his writing on the experience of art. For Dewey, the fusion of separate elements into unity, while simultaneously enhancing their identity, constitutes *an* experience, something that is associated with fulfillment and satisfaction. A proper work of art is an important example of *an* experience. *An* experience is not exclusively emotional, practical or intellectual, but

related to the development of an idea, the fulfillment of which constitutes *an* experience.

Notably for the programming perspective, Dewey associates an aesthetic quality with thinking, and believes the experience of thinking provides emotional satisfaction (Leddy, 2016). The idea of consummation and fulfillment as a result of practical action resonates with multiple crafts, including the Zen arts. For Dewey, intellectual and skilled pursuits are aesthetic by default.

Leddy (2016) describes the structure of Dewey's "*an* experience" thus:

> The subject undergoes something or some properties, these properties determine his or her doing something, and the process continues until the self and the object are mutually adapted, ending with felt harmony. This even holds for the thinker interacting with his or her ideas. When the doing and undergoing are joined in perception they gain meaning. Meaning, in turn, is given depth through incorporating past experience.

This idea has a direct parallel with the design and implementation of a computer program. Particularily interesting is the articulation of the emotional fulfillment that comes with the conclusion of such a project – for the thinker interacting with his or her ideas, which is an apt description of a programmer.

Similarly resonant is Dewey's idea of expression. "Impulsion" is the aspiration of a total organism to develop in response to pressure from environmental interaction. Dewey counts various external objects, such as tools and culture, as belonging to the total organism. The organism depends on the environment to surive: the essential external objects must be acquired. The opposition that the environment presents to the organism's impulsion must be overcome, which results in purpose and meaning: the result is elation. This stimulates reflective action, transforming the impulsion into a medium for creativity (Leddy, 2016).

Wittgenstein: Connections

In contrast to earlier thinking on beauty, Wittgenstein criticized the idea of beauty as a property belonging to an object - as a linguistic sign carrying an intrinsic meaning. He proposes activities rather than descriptions as the primary expression of aesthetics: a tailor's work is appreciated not by describing the suit but rather by wearing it (Hagberg, 2014).

Wittgenstein discusses the concept of correctness in musical context: rule-learning by drills in arts like counterpoint and harmony enable the

student to undestand the proper interpretation of aesthetic rules in any particular context (Hagberg, 2014). This reflects the idea that action is primary and that actual knowledge is encoded in complex, interconnected systems rather than the words of a cultural-contemporary "language game".

According to Hagberg (2014), Wittgenstein's aesthetic judgement is culture-dependent, and "both more immediate and expansive than a simple mechanistic account could accommodate". The exact conditions for achieving aesthetic satisfaction are impossible to codify.

Wittgenstein's aesthetics emphasize the import of making connections and associations – "connective analysis". It is the kind of holistic, systemic understanding that thinking based on linguistic signs and reduced meanings will likely miss. This has implications for computer system design: if we prescribe to Wittgenstein, the beauty or lack thereof in a software system can only be determined from its totality – the innumerable connections between all its parts.

Mathematical Beauty

Mathematical beauty and logical elegance could be considered the underpinnings of the normative aesthetics of computer science. Some properties that are sometimes understood as beautiful in mathematics include astonishment – how mathematical proofs can introduce unforeseen "worlds" by strange leaps of intuition, only afterwards submitting to rigour. For Burke, astonishment is a self-preserving property, the foundation of the sublime, of grand, forbidding objects inspiring near-pious awe (Shelley, 2014). In software, these adjectives some of the most complicated, critical and depended-upon systems, such as operating systems, garbage collectors, virtual machines and compilers.

Mathematical beauty is also associated with deep connections. Fundamental theorems, such as Euler's formula that links trigonometry and logarithms, is often cited as beautiful. Such connective beauty, indicative of deeper truths and systemic organization, recalls Wittgenstein's aesthetics (Hagberg, 2014). Deep connections abound in computer science: functional programming, originating in the λ-calculus, and object oriented programming, arguably an extension of the Turing model, exhibit a surprising mutual correspondence: the object–closure equivalence, dealing with encapsulation and scoping of data, symbols and names.

Beauty and elegance are sometimes related to brevity. In philosophy, Occam's razor is the conjecture that when two alternative explanations for a phenomenon exist, in the absence of other information, the simple explanation is more likely to be correct. This statement concerns truth values; in mathematics, simplicity is primarily aesthetic. Most mathematicians would

likely agree that given two alternative formulations for a theorem, the simpler one would more likely be beautiful. Simple computer programs are also widely appreciated, especially if they accomplish a surprisingly involved task.

Paul Dirac was a proponent of mathematical beauty, quoted as saying that he prefers a beautiful theorem to a true one (Stewart, 2007, pp. 277-279). While most mathematicians would likely require logical rigour and consistency from the theorems they consider beautiful, Dirac's argument is a deeper one: he believed that the true laws of nature had to be beautiful, and beauty acts as a clue for humanity in the search for deepest truths. Beauty as a form of divine or transcendental guidance goes back all the way to the British internal-sense theories of Shaftesbury and Hutcheson (Shelley, 2014).

Beauty in Programs

If mathematical beauty is sometimes considered as a guide to deep truths of nature, programmatic beauty could be taken as an indication that the program in question approach a "truth" or optimal arrangement from a particular point of view. A beautiful program could be easy to read or write or perhaps maintain in the future – this beauty could be explained in the terms of the British imagination theories and absence of imagined pain (Shelley, 2014).

Many philosophers have striven to put to words the experience of beauty. Sometimes, the words themselves could be argued to more closely fit crafts such as programming – a natural fusion of reason and creativity. Programming is not "solved", there is plenty of room for creativity and intuition. Software systems are large, interconnected and difficult to understand fully, true Wittgensteinian constructs (Hagberg, 2014). This is especially true as they grow in scope and features are added.

Aesthetics can serve as an early indication of a program design going *wrong*. Programs are built as layers upon layers of abstraction, and the mistakes and imperfections on each layer cascade and accumulate superlinearly. Aesthetics can guide software design as beauty guided Dirac in his search for laws of nature (Stewart, 2007). Dirac did not believe that beauty and truth are one and the same – he believed that beauty is necessary for truth. However, as Huxley said (in Stewart 2007), "Science is organized by common sense, where many a beautiful theory can be killed by an ugly fact." This is especially true in computer science, where execution, hardware, users and I/O embody grisly realism in contrast to the pure bliss of reason.

Programming is indeed a discipline where advanced practitioners recog-

nize an occasional need for an "ugly hack". Yet, such hacks often relate to highly tuned, well-performing programs: beautiful in the utilitarian sense that Wolff describes (Guyer, 2014). As a counterexample, many excellent textbooks and tutorials feature computer programs that are beautifully simple, distinct and clear – yet unusable in practice due to performing particularily badly in terms of execution time or space. These programs exemplify programming as communication; an aspect of programming with its own distinct concerns.

References

Backus, J. (1979). The History of FORTRAN I, II, and III. *Annals of the History of Computing*, 1(1):21–37.

Clark, L. B., Wilson, R. G., and Robert, C. (2000). *Comparative programming languages*. Addison Wesley, Harlow, 3rd edition.

Dijkstra, E. W. (1968). Go To Statement Considered Harmful. *Communications of the ACM*, 11(3):147–148.

Doy, G. (2008). Books and catalogues in brief : "Practice as research : approaches to creative arts enquiry," edited by Estelle Barrett and Barbara Bolt. *Art Book*, 15(3):78–79.

Freeth, T. (2006). Decoding the Ancient Greek Astronomical Calculator Known as the Antikythera Mechanism. *Nature*, 444:587–591.

Guyer, P. (2014). 18th Century German Aesthetics. In Zalta, E., editor, *The Stanford Encyclopedia of Philosophy2*. Fall 2014 edition.

Hagberg, G. (2014). Wittgenstein's Aesthetics. In Zalta, E., editor, *The Stanford Encyclopedia of Philosophy2*. Fall 2014 edition.

Halacy, D. S. (1970). *Charles Babbage, Father of the Computer*. Crowell-Collier Press.

Hodges, A. (2012). *Alan Turing: The Enigma*. Princeton University Press, centenary edition.

Ifrah, G. (2001). *The Universal History of Computing: From the Abacus to the Quantum Computer*. John Wiley & Sons,Inc., New York.

Johnstone, A. (2014). Babbage's Language of Thought.

Kearney, H. F. (1971). *Science and Change, 1500-1700*. Littlehampton Book Services Ltd.

Kleene, S. C. (1981). Origins of Recursive Function Theory. *Annals of the History of Computing*, 3(1):52–67.

Leddy, T. (2016). Dewey's Aesthetics. In Zalta, E., editor, *The Stanford Encyclopedia of Philosophy2*. Spring 201 edition.

Shelley, J. (2014). 18th Century British Aesthetics. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Fall 2014 edition.

Steele, G. L. (1984). *Common Lisp*, volume 2. Digital Press.

Stewart, I. (2007). *Why Beauty is Truth*. Perseus Books, New York.