ON THE EMBODIED AESTHETICS OF CODE

Scott Dexter, Melissa Dolese,

Angelika Seidel, Aaron Kozbelt

Finally, a computer program only has one meaning: what it does. . . . Its entire meaning is its function.
Ellen Ullman (Rosenberger, 1997)

I wouldn't compare a program with the Mona Lisa, but it does have a simplicity and elegance that's quite handsome. Stylistic distinctions of different programs are intriguing, very much like the differences art critics might see between Leonardo's Mona Lisa and a Van Gogh ... When you write an algorithm using M expressions, it's so beautiful you almost feel it could be framed and hung on a wall.
Gary Kildall (Lammers, 1986/2006: 64)

If Ellen Ullman – a programmer – is right, then how do we make sense of reports from other programmers – such as Gary Kildall – on the aesthetic value of the code they read and write? Of course, a significant aspect of the meaning of code derives from the human interpretation of the computations being carried out when the code executes: 'an internal representation is merely the potential for what may be manifest in the external representation' (Laurel, 1998: 46). But code – source code – itself carries rich representations and meanings to those who know how to read and write it.

The artifacts designed by programmers are not material objects; at most they are abstractions capturing some desired essence of their material analogs. But the habits of thought of these programmers are habits first cultivated through embodied experience in the material world. As Christopher Kelty characterizes them, 'Geeks live in specific ways in time and space. They are not just users of technology, or a "network society," or a "virtual community," but embodied and imagining actors' (Kelty, 2008: 77). It is this embodiment, the specific ways of living in time and space common to all humans, which ultimately provides the meanings – functional and aesthetic alike – of source code.

In this paper we consider anecdotal and empirical evidence bearing on the aesthetics of programming, placing these in dialog with accounts of the embodied experience of programming and recent studies of significance of the embodiment in the production of meaning. We identify code, aesthetics, and embodiment as a fertile nexus which may shed light on the nature of each and on the relations among them. We close with a discussion of directions for research which rely on empirical methodologies to inform the aesthetic and embodied properties of code.

Code and aesthetics

Among both laypersons and scholars, aesthetics – as a mode of experience or as a domain of intellectual inquiry – is usually associated with the fine arts, rather than the sciences or technical disciplines like software development. This bias, echoing C. P. Snow's (1960) famed 'two cultures,' remains evident both in comprehensive accounts of scientific creativity (e.g., Feist, 2006; Simonton, 2004), which hardly mention aesthetics, and in mainstream research on aesthetics, where historically scholars have focused on the fine arts (e.g., Leder *et al.*, 2004; Levinson, 2003).

However, aesthetic experience, judgment, and preference need not be confined to artistic, musical, or literary artifacts. Indeed, a number of scholars have strongly suggested that aesthetic issues apply just as well to scientific and technical domains (e.g., Curtin, 1982; Tauber, 1996; Wechsler, 1977), though this connection remains largely unexplored through any rigorous methodology. Much of the evidence in support of this claim comes from the first-person reports of eminent mathematicians, scientists, and technologists themselves. For instance, Hadamard (1954) collected a number of such accounts from mathematicians, including eminent figures such as Henri Poincaré, and argued from them that the roots of creativity involve many unconscious processes, including the aesthetically-based selection of ideas. The pioneering neurologist Santiago Ramón y Cajal likewise emphasized this point, writing that in neurology his 'aesthetic instincts found full satisfaction at last' (Ramón y Cajal, 1937/1989: 363). Eminent physicists such as Paul Dirac (see Kragh & Hovis, 1993), Werner Heisenberg (1974) and Subrahmanyan Chandrasekhar (1990) have also written on the significance of beauty in doing physics. Chandrasekhar suggested that aesthetic concerns may be a primary motivating factor for scientists' continued activity as well as a basis of theory choice:

> It is, indeed, an incredible fact that what the human mind, at its deepest and most profound, perceives as beautiful finds its realization in external nature. What is intelligible is also beautiful. We may well ask: how does it happen that beauty in the exact sciences becomes recognizable even before it is understood in detail and before it can be rationally demonstrated? In what does this power of illumination consist? (Chandrasekhar, 1990: 66)

Root-Bernstein (2002) makes a stronger claim, asserted that all human inventions, stemming from science, engineering, or mathematics, have the potential to evoke aesthetic responses that are the same as those evoked by the arts, and that the drive to experience beauty has often resulted in great scientific research.

An emphasis on aesthetics also strongly characterizes many computer programmers' and software developers' accounts of their domain. As in other scientific domains, anecdotal first-person accounts (e.g., collections by Lammers, 1986/2006; Oram & Wilson, 2007) have been the main source of evidence bearing on this issue. Interestingly, programmers' own descriptions of the role of aesthetics in software are close in character to characterizations from fine arts domains and, indeed, often cite such domains as direct analogies. Charles Simonyi, who oversaw the development of flagship Microsoft products such as Office and Excel, observed:

> Some people have different opinions about what makes the structure [of a program] beautiful. There are purists who think only structured programming with certain very simple constructions, used in a very strict mathematical fashion, is beautiful. . . . But to me, programs can be beautiful even if they do not follow those concepts if they have other redeeming features. It's like comparing modern poetry with classical poetry. (Lammers, 1986/2006: 13)

Similarly, Brian Kernighan, one of the designers of the programming language AWK, noted the analogous relationship between writing program code and writing English prose:

> In both text and programs, I tend to work over the material many times until it feels right. There's a lot more of this in prose, of course, but it's the same desire, to have the words or the code be as clear and clean as possible. (Biancuzzi & Warden, 2009: 118)

C. Wayne Ratliff, who designed and managed the dBASE series of database systems, emphasized the diagnostic value of balance, which echoes the notion of compositional balance in painting (Arnheim, 1988; Locher & Nagy, 1996), and his comments suggest a connection between aesthetic judgment and functional assessment:

> If you write a program well, it's very elegant; it sings, it's well built. I enjoy it from an engineering point of view, just like a well-built car, a well-built bridge, or a well-built building. Everything about it seems in balance, tuned. . . . When things get really out of balance, you know something is wrong. There's probably some inherent fault that makes it out of balance. (Lammers, 1986/2006: 120)

Such anecdotal reports, which represent a thread running through many programmers' reflections on their craft, reinforce the need to better understand and integrate aesthetic considerations into the creative activity of scientific and technical domains. These accounts may also be connected to perspectives from the philosophy of science, such as that of James McAllister (1996), who suggested that any property of a scientific theory might be regarded as aesthetic 'if scientists in the relevant disciplines react to [the property] publicly as aesthetic, for example by . . . applying to it standard terms of aesthetic appreciation, such as "beautiful," "elegant," "pleasing," or "ugly"' (McAllister, 1996: 36).

McAllister further argued that the doctrine, espoused by several prominent scientists such as Chandrasekhar and Heisenberg, that beauty is an attribute of truth implies an agreement between an entity's perceptual aspects and its utilitarian qualities. That is, the beauty of a theory may evidence its proximity to the truth, and therefore aesthetic criteria may effectively indicate scientific utility. But ideas of scientific utility (particularly the productive potential of a theory) evolve, and aesthetic canons must follow suit, in what McAllister terms 'aesthetic induction'. This echoes the response of architecture or industrial design, for example, to concerns of utilitarian performance. In both cases, the demonstrated practical worth of a work – the empirical success of a scientific theory or the utility of a building – can contribute to reshaping the basis of aesthetic canons on which subsequent contributions are evaluated.

Davies (2006) framed the connection between aesthetics and functionality similarly, positing that for any utilitarian object with aesthetic features that are not trivial or incidental, aesthetic appraisal is related to that object's function. An object is 'functionally beautiful' to the extent that its aesthetic properties contribute to its overall performance – the functional beauty of an object enhances its fulfilling its primary function. So, for example, a 'beautiful chair is one having features that make it graceful and stylish and, at the same time, comfortable to sit on, stable and supportive of the back' (2006: 237). These views resonate with Kildall's suggestion that, in code, not only does functionality reside in the brute performance of a program, but also, in a fundamental way, in the aesthetic dimensions of the program that enable it to be appreciated, repaired, or modified by other programmers.

Based on arguments from the philosophy of science as well as firsthand accounts by scientists and technologists themselves, there are reasons to believe that the products of science and technology, including software code, can exhibit aesthetic properties and induce aesthetic experiences. However, quoting Chandrasekhar or Dirac among physicists, or Ratliff or Simonyi among software developers, does not necessarily reveal the extent to which aesthetic considerations are a pervasive aspect of scientific or technical thinking. Perhaps aesthetic aspects of science are mainly characteristic of only the greatest practitioners, perhaps at the moments of their greatest discoveries. Alternatively, aesthetic concerns may be relatively common in scientific or technical domains, at least among those with the requisite expertise to appreciate potential instances of beauty in a domain.

However, some initial quantitative research appears to confirm that basic claims about the importance of aesthetics, such as those made by programmers above, appear to be fairly representative of programmers' experiences. A recent empirical investigation

(Kozbelt *et al.*, 2010) addressed the issues of the frequency, nature, time course, and judgment criteria of aesthetic experience among 50 software developers with varying levels of experience. These programmers reported having fairly frequent aesthetic experiences with code, though somewhat less often and intensely than with other creative artifacts, such as paintings. Overall, participants reported that in their experience, judgments of 'ugly' code were made faster than those of 'beautiful' code, which in turn were made faster than those of 'correct' code. Aesthetic considerations of code were rated as quite important, though not as important as functionality. Finally, judgments of the relative importance of various aesthetic judgment criteria were highly correlated among experts and novices alike. These results corroborate many of the anecdotal claims made about aesthetics and code in the software literature. Moreover, this study suggests that a quantitative approach to studying aesthetics and code is a fruitful research direction, with potentially trans-domain implications for aesthetics and creativity.

Code and embodiment

We now turn to the relation between code and the notion of embodiment, which, we argue, is a relation central to the full understanding of code's meaning, and one historically undervalued. Programmers know all about the sensation of disembodiment:

> Among the five assembler programmers of the project team there was one who one night sat down at the terminal, got glassy eyes, and slipped into a mental state in which he could not be talked to any more. . . . Among his colleagues the man was called the 'trance programmer.' He once commented . . . 'You could fire a cannon next to me and it would not bother me.' (Molzberger, 1983: 247)

This is but an extreme example of a phenomenon familiar to any seasoned programmer – the intense mental focus on a complex and evolving abstraction which results in a temporary experience of something akin to disembodiment.[1] Presumably, it is the desire to maintain this state for as long as possible which leads programmers to such apocryphal acts of bodily neglect as subsisting on pizza and soda, deferring routine hygiene, and sleeping under their desks. (Conversely, programmers' employers, perhaps most famously

Google, are noted and lauded for making it possible for programmers to tend to (most of) the needs arising from embodiment, ranging from massage to laundry, while at work.)

Yet in the context of software development, the question of embodiment is not just an irritating distraction. From the earliest days of machine computation, programmers and philosophers alike have speculated about the transhumanist possibility of superior intelligences freed from bodies as we know them—whether in the form of autonomous machine intelligence or in the form of some sort of chassis into which we could upload our individual intelligences. The question of whether this is possible haunts many discussions of programmer culture. Kelty's (2008) *Two Bits: The Cultural Significance of Free Software*, for example, is primarily a study of, indeed, the cultural significance of free software. Yet, as Kelty explains, he finds 'certain aspects of transhumanism are present across the spectrum of engineers, scientists, and geeks' (321). That is, while few programmers may ardently anticipate a Great Uploading, transhumanist beliefs that technological interventions may and should destabilize culture and politics are 'widespread among technically adept individuals' (87). Mitch Kapor, a technologist and investor perhaps best known as the founder of Lotus Corporation, is a notable exception. He explains his pessimism about the prospect of human-level machine intelligence:

> As humans. . . we are embodied creatures; our physicality grounds us and defines our existence in a myriad of ways. . . . Emotion is as or more basic than cognition; feelings, gross and subtle, bound and shape the envelope of what is thinkable. . . . When I contemplate human beings in this way, it becomes extremely difficult even to imagine what it would mean for a computer to perform [being human]. (Kapor, 2002)

Some of the most compelling metaphors for the practice of programming itself arise from this embodiment, this physicality. Veteran programmer and poet Richard Gabriel has suggested 'habitability' as a major design goal for any programming project: 'Habitability makes a place livable, like home. And this is what we want in software—that developers feel at home, can place their hands on any item without having to think deeply about where it is. It's something like clarity, but clarity is too hard to come by' (Gabriel, 1996: 11). While this notion appears at first to be just another analogy between software and architecture (indeed, Gabriel draws no little inspiration from the work of architect Christopher Alexander), it is more: it is an assertion that programming is itself a profoundly embodied practice, that the process of designing and building a program is informed by our embodied experience with and within other, materially instantiated, functional designs. This perspective continues to drive research into software engineering tools and techniques, such as the work of Wettel & Lanza, who have developed a '3D visualization of software systems hinging on the city metaphor' (2007: 1).

More generally, programmers' own accounts of their activities and experiences (e.g., Lammers, 1986/2006; Oram & Wilson, 2007) are replete with terms like 'balance,' 'flow,' 'natural,' and 'flexible,' which are terms based, however unconsciously, on human embodied reality. That is, these accounts are drawn, ultimately, from the fact that human beings have material bodies that move through time and space. This is a generally well-known fact, though one historically discounted in studies not only of programming but of human cognition construed most broadly. Recent scholarship on embodiment has shown that it has enormous implications for the nature of human thinking and bears on many venerable philosophical issues (Lakoff & Johnson, 1999). The modern study of embodied cognition can be traced back to scholars such as Bourdieu (1977) and Mauss (1979), who described the ways values become 'embodied'; since then, embodiment has become a major theme contemporary psychology and cognitive science. Empirical research on embodiment has revealed its substantial impact on mental processes, including perception, memory, language, emotion, and social cognition (see, e.g., Boroditsky & Ramscar, 2002; Markman & Brendl, 2005; Niedenthal *et al.*, 2005). The demonstrated power of this framework suggests that it may be fruitfully applied in analysing a variety of domains, including some – like mathematics and computer science – in which embodiment has been held to be at best an irrelevance and at worst a nuisance (e.g., Kapor, 2002).

The embodied foundations of programming share much with the embodied foundations of mathematics: 'Our mathematics of calculation and the notation we do it in is chosen for bodily reasons . . . [but] the algorithm, being freed from meaning and understanding, can be implemented in a physical machine called a computer, a machine that can calculate everything perfectly without understanding anything at all' (Lakoff & Nuñez, 2000: 86). That is, while an implementation of an algorithm may be perceived at some level as being simply rote calculation, its grounding in meaning arises from human embodiment.

Lakoff and Nuñez, in their treatment of the embodied origin of mathematics, describe a 'Romance of Mathematics,' which they seek to dispel, characterizing it as 'not a story with a wholly positive effect' (2000: 340). Among the premises they ascribe to this mythology are 'Mathematics is abstract and disembodied – yet it is real' (xv) and 'Mathematics has an objective existence . . . independent of and transcending the existence of human beings or any beings at all' (xv). From these and other premises, they suggest, it is natural to conclude that 'Because mathematics is disembodied and reason is a form of mathematical logic, reason itself is disembodied. Hence, machines can, in principle, think' (xv).

That is to say, from the Romance of Mathematics we may derive a similarly flawed 'Romance of Computation,' in which computation exists outside of and independent from human experience, and substantially orders the universe in ways beyond our ken and control. Yet, as Lakoff and Nuñez demonstrate in great detail, there is no evidence (nor can there be any) for a truly disembodied mathematics.

Indeed, the early tradition of cognitive science was based exactly on such a Romance of Computation. As Mark Johnson describes it:

> For classical cognitive science, it is assumed that cognition consists of the application of universal logical and formal rules that govern the manipulation of 'internal' mental symbols, symbols that are supposedly capable of representing states of affairs in the 'external' world. . . . The internal/external split that underlies this view presupposes that [the internal language of thought] could be detached from the nature and functioning of specific bodily organisms, from the environments they inhabit, and from the problems that provoke cognition. Given this view it would follow that cognition could take place in any number of suitable media, such as a human brain or a computing machine. This theoretical viewpoint was instrumental in the development of the first electronic calculating

> machines and general-purpose computers. (Johnson, 2007: 119)

Johnson goes on to argue that even logical inference is grounded, through metaphor, in human embodiment:

> [T]he logic of our bodily experience provides all the logic we need in order to perform every rational inference, even with the most abstract concepts. In our metaphor-based reasoning, the inferences are carried out according to the corporeal logic of our sensorimotor capacities, and then, via [a] source-to-target mapping, the corresponding logical inferences are drawn in the target domain. (179)

Embodiment and the modern conception of computation are in a continually productive state of mutual destabilization. Alan Turing's originary formulation of a computing machine makes a metaphoric leap from the embodied action of humans manually performing calculations to an abstracted mechanical process.[2] The resulting mantra that cognition is computation, promulgated by early cognitive science researchers, defined thought in exactly those terms that could be instantiated by the digital computer, downplaying significant other aspects of the human mind, such as motivation, emotion, and cross-cultural differences (e.g., Gardner, 1987). Because the operations of these machines have no apparent meaning outside of human activity, it is easy to conclude, as Ellen Ullman did, that the meaning of a program is identical to the human-interpreted result of its function: once the 'internal' mental symbols are brought outside the body, the crucial connection with 'external' states is severed, the 'internal' symbols all but vanish as they are stripped of their meaning, and we are left with machines which, with a compelling illusion of near-autonomy, traverse a wide range of meaning-laden states. To begin to repair this flawed scheme and recover some meanings of code, we must identify and develop some new formulation for the embodied meaning of computation, perhaps echoing the historical trajectory of the metaphors of software from being based closely on the machinic 'body' to those derived from human embodied experience.

From early programs which directly specified how the knobs and switches on the computing machine should be set, to more advanced techniques allowing programmers to write programs as sequences of 0s and 1s (using symbolic notation, which could be stored and retrieved by the machine, to represent machine configurations), to contemporary techniques intended increasingly to permit the description of computations in 'human-readable' terms, a driving force in the evolution of programming has been the creation of tools, techniques, and computing hardware which permit programmers to be increasingly ignorant of the material realities of the machine, focusing instead on the abstractions they create and manipulate. As Bjarne Stroustrop, designer of the C++ language, says:

> 'close to the hardware' means that the model of computation is that of the computer—sequences of objects in memory and operations as defined on objects of fixed size—rather than some mathematical abstraction. That is true for both C++ and Java, but not for functional languages.[3] . . . The real problem is how to get from the human conception of problems and solutions to the machine's limited world. You can 'ignore' the human concerns and end up with machine code . . . . You can ignore the machine and come up with a beautiful abstraction that can do anything at extraordinary cost and/or lack of intellectual rigor (Biancuzzi & Warden, 2009: 5).

At its root, software is a supreme act of metaphor: the manipulation of abstractions in contemporary programs is guided and governed by interleaved layers of metaphor. The design of the C language, for instance, encourages programmers to think of their programs as sequences of instructions which store, retrieve, and manipulate values stored within a homogeneous linear sequence of storage cells. The LISP programming language encourages programmers to think of programs simply (yet amazingly powerfully) as lists of values; seasoned LISP programmers in fact use LISP to create *ad hoc* languages based, as closely as they wish, on the particular metaphors informing their problem domain. Programs in Prolog are collections of 'facts' and 'rules' which are 'queried' to provoke a computation. The 'object-oriented' programming paradigm asks programmers to organize their programs as collections of interacting 'objects' (which bear some metaphoric resemblance to physical objects). These metaphors do not simply provide a way to think about programming, but tend to structure programmers' thought about the computational process itself. As Robin Milner, designer of the

ML language, observed, '[S]ome languages . . . actually influence the way that the programmer thinks about the task. Object-oriented languages have done very well from this viewpoint, because the notion of object helps to clarify thought in a remarkable variety of applications' (Biancuzzi & Warden, 2009: 213).

As Johnson shows, such metaphors, even though they operate at a very high level of abstraction, are rooted in the neural structures which attend to the sensorimotor aspects of our embodied experience. In the balance of this paper, we aim to establish some connections between the embodied roots of programming and the embodied roots of aesthetic meaning.

Embodiment and aesthetics

The aesthetic meanings of code are demonstrably significant to programmers: the aesthetic qualities of code are popular topics of discussions of code both online and in traditionally published work; adherents of different programming languages have been known to contend, sometimes quite aggressively, to show the aesthetic superiority of their chosen language (e.g, '[emacs] is written in LISP, which is the only computer language that is beautiful' (Stephenson, 1999: 96)); as we argued above, there is some reason to believe that such aesthetic judgments are components of the various assessments programmers make throughout the development process. Embodiment, too, plays a foundational if vexed role in determining the meaning of computation, and of code in particular. In this section, we review recent scholarship on the grounding of aesthetics in embodiment more generally, with a view toward outlining possible ways of empirically exploring the relations among these (and other) constructs in software development.

The most sophisticated aesthetic judgments of code are made within the metaphoric frameworks which define the language in which the code is written. Sometimes, aesthetic judgments of code are targeted especially at the algorithm being implemented.[4] Or, aesthetic judgments may be targeted at stylistic concerns, of textual formatting, naming conventions, or documentation. But the most intriguing judgments have to do with the choices made with respect to the organizing metaphor(s) in play.

A C program might make especially clever or efficient use of the underlying machine model, as in Warren's treatment of the 'fundamental' yet 'deceptively simple' operation of counting the number of a memory cell's bits which have the value 1, an operation known as 'population count' (Warren, 2007: 147). Warren considers a range of algorithms and their implementations, explicitly assuming a machine model which 'has the fundamental instructions generally found on a RISC or CISC computer: shift, add, and, load, conditional branch, and so forth' (147). The solutions he considers have, he asserts, 'some beauty to an eye that values efficiency, conciseness, and useful cleverness' (149), though, as he notes, programs that are *too* tightly dependent on the specifics of the machine may lose some aesthetic value. Warren shows that one algorithm may be expressed in C code as

```
x = (x & 0x55555555) + ((x >> 1) & 0x55555555);
x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
x = (x & 0x0F0F0F0F) + ((x >> 4) & x0F0F0F0F);
x = (x & 0x00FF00FF) + ((x >> 8) & 0x00FF00FF);
x = (x & 0x0000FFFF) + ((x >> 16) & 0x0000FFFF);
```

but immediately offers the 'simplification'

```
int pop (unsigned x) {
    x = x – ((x >> 1) & 0x55555555);
    x = (x & 0x33333333) + ((x >> 2) & 0x33333333);
    x = (x + (x >> 4) & 0x0F0F0F0F;
    x = x + (x >> 8);
    x = x + (x >> 16);
    return x & 0x0000003F;
}
```

observing 'Unfortunately, the [second implementation] has lost most of the regularity and elegance of the code from which it was derived. A consequence of this is that it is no longer immediately clear how to extend the code to a 64-bit machine. But it's hard to pass up all those opportunities to save instructions!' (150-151).

An object-oriented program might rest on an especially apt framework of objects, as in Otte & Schmidt's (2007) C++ code for a networked logging service. In this case, 'the beauty of [the] solution stems from its use of patterns and [object-oriented] techniques to balance key domain forces, such as reusability, extensibility, and performance. In particular, [this] approach enables developers to identify common design/programming artifacts, [and] . . . also provide[s] a mean to encapsulate variabilities in a common and parameterizable way' (431). That is, the beauty of their approach inheres in the way object-oriented metaphors are marshaled explicitly to conceal the specifics of a particular machine. '[O]bject-oriented languages . . . combined with *patterns* (such as Wrapper

Facades, Adapters and the Template Method), and frameworks (such as host infrastructure middleware like ACE and the Java class libraries for network programming) . . . mask syntactic and semantic differences between platforms . . . [allowing developers to avoid] wrestling with the accidental complexities of programming the low-level networking and O[perating] S[ystem] infrastructure' (430). Some of these advantages are demonstrated in this fragment of code:

```
    template <typename ACCEPTOR, typename MUTEX> void
    Logging_Server<ACCEPTOR, MUTEX>::run (void) {
     try {
                // Step 1: initialize an IPC factory endpoint to listen for
                // new connections on the server address.
                open ( );

                // Step 2: Go into an event loop
                for (;;) {
                            // Step 2a: wait for new connections or log records
                            // to arrive.

                            wait_for_multiple_events ( );

                            // Step 2b: accept a new connection (if available)

                            handle_connections ( );

                            // Step 2c: process received log record (if available)

                            handle_data ( );
                }
     } catch (...) { /* ... Handle the exception ... */ }
}
```

Beauty inheres specifically in this code (and the code it invokes or structurally prefigures) because of both the 'pattern-based design' for handling 'variation in concurrency models' by 'providing specific implementations' of methods such as wait_for_multiple_events ( ) and the 'template-based design' for handling 'variation in . . . synchronization mechanisms' by 'plugging different types into the ACCEPTOR and MUTEX template parameters' (439).

Or, a LISP program may make effective use of its broad facilities for manipulating language. Dybvig (2007) explicates and extends a 'hygienic macro expansion algorithm', which he characterizes as 'both clever and elegant' (414). This algorithm is a technique for allowing LISP programs to manipulate LISP source code, so, 'the most important aspect of the [extended] mechanism is its abstract representation of program source code as *syntax objects*' (414). That is, the code Dybvig writes about is informed by LISP's deep affinity for the formal aspects of language, specifically, the especially porous boundary between the language of the code being written and the language of the syntactic objects being manipulated by the code. The depth of the metaphor of language in this example (which is characteristic more generally of LISP's strengths) means that the particular beauty of the code implementing this algorithm does not leap from the page straight into the eye of the non-LISP-fluent reader. However, this is no constraint on its capacity to elicit aesthetic responses, Dybvig suggests, as 'there can still be beauty in complex software as long as it is well structured and does what it is intended to do' (428).

This brief foray into some actual source code adds weight to our claim that source code is shot through with metaphor, whether the metaphor of the computing machine; metaphors of objects, patterns, and frameworks based on concepts like 'wrapper,' 'facade', and 'template'; or the metaphor of formal language itself; and that these metaphors are both grounded in embodiment and themselves ground aesthetic responses to code.

This returns us to our fundamental concerns about the interplay among code, aesthetics, and embodiment. The pervasiveness of aesthetic experience across a broad range of domains, modalities, and kinds of human and natural artifacts (see, e.g., Saito, 2008) suggests a very general underpinning for human aesthetics, one whose certain identification lies well beyond the compass of this paper. But there are possible answers. For instance, the emerging domain of 'neuroaesthetics' argues that aesthetic experience may be fruitfully understood in terms of the capacity of, say, works of art, to stimulate (or titillate) the mind in a way that evokes aesthetic pleasure at the level of brain structures and processes (see Kozbelt, in press); this would be one means of grounding aesthetic experience in a more or less universal terms (as opposed to those which are strongly culturally determined).

However, thinking more broadly than just the nervous system, the notion of embodiment described earlier may serve as a complementary means of fundamentally grounding aesthetic experience. Johnson (2007) argued that everything we deem aesthetic involves an experience in which we have the capacity to make and experience meaning, and that we do so through our visceral, embodied connections with the world. Through bodily perceptions, movements through space, and emotion it becomes possible for meaning to be formed and for aesthetic experiences to occur. In fact, meaning and aesthetic experience may be more closely identified than we suspect: '[T]he structures processes, and qualities that make art possible and valuable are exactly the same ones that constitute *all* meaning, thought, and understanding. . . . [T]hese aspects of embodied meaning are not, for the most part propositional, and it therefore follows that meaning cannot be primarily linguaform and propositional' (Johnson, 2007: 213).

In other words, beauty is not just a characteristic of an object; it is a felt experience. This may play out in all manner of domains and modalities. For instance, Johnson (2007) proposed that music is a 'presentation and enactment of felt experience' (238): the tension that is felt in music is possible through the listener's engagement as they experience their own bodily sensations of tension. The identification of a creative artifact or an everyday happening as eliciting or enacting a felt experience could warrant an exclamation of beauty. Even negative responses, such as anger and disgust (Silvia & Brown, 2007), qualify as aesthetic due to their embodied nature: the valuation of an object as disgusting is felt as a gustatory response in the beholder.

Code may appear to some to be among the *most* 'linguaform and propositional' modes of contemporary human expression and, thus, completely unsuitable for attaching completely different forms of meaning. But, as we have shown, the development of modern programming depends absolutely on a complex scaffolding of metaphor and non-propositional meaning drawn from the roots of embodied human experience. It is this accretion of meanings that form the basis, and provide the significance, of aesthetic judgments of code.

Synergies

Aesthetics, code, and embodiment synergistically interact in fundamental ways, which are far from fully understood. To our knowledge, theoretical discussions of the three-fold relations between aesthetics, code, and embodiment remain rather embryonic to date; empirical research is even scarcer. Pursuit of the nature of these manifold interactions will, we expect, yield not only specific results informing the nature of this intersection but also broader and richer accounts of all three concepts.

The functional role of aesthetics plays out most richly when contextualized as part of the creative process of software development. For instance, the rather interesting empirical finding that programmers report being more quickly able to discern beauty than correctness (Kozbelt *et al.*, 2010) suggests that aesthetic-laden evaluative processes may drive judgment and decision making about software code – potentially both as final products and works-in-progress. Specifically, a creative process geared toward high efficiency could well emphasize more intuitive, affective, aesthetically-motivated judgment criteria over more logical, conscious, attention-demanding strategies (see also Perkins, 1981). Indeed, since Kozbelt et al found that judgments of ugly code were reportedly made even faster than those of beautiful code, this aesthetic mode may be particularly useful for detecting problems in a program.

The links between aesthetics and creativity are both intuitive and well-appreciated. In contrast, our understanding of the relationship between embodiment and creativity remains rather underdeveloped; however, there are prospects for fruitful dialogue. In terms of forging connections between these two literatures, probably the most direct approach is via the nature of metaphor, which has strong traditional links to the notion of embodiment (e.g., Lakoff & Johnson, 1980); moreover, in the creativity literature, metaphor is often discussed as a fundamental mechanism of creative thought, together with processes like conceptual combination, conceptual expansion, and mental imagery (Ward et al, 1997). In a related vein, Gibbs (2005) describes instances of scientific and artistic creativity stemming from kinesthetic imagery and other embodied sources.

Methodologically, the most likely means of gaining evidence bearing on all of these issues is the technique of concurrent verbal protocol analysis (Ericsson & Simon, 1984), in which persons verbalize their conscious thoughts, without undue interpretation, as they work to solve some problem in the laboratory. Typically, the verbalizations are recorded, transcribed, parsed, and coded for the frequency of various categories of statements of interest. For instance, the kind of language used by programmers in edited collections (e.g., Lammers, 1986/2006; Oram & Wilson, 2007) reflecting aesthetic or embodiment or metaphoric themes could be readily used to construct a coding system for such statements. Contextualized in a creative problem solving task, such as editing poorly-written or ugly code, this would provide a dynamic window on the cognitive processes involved in the creation of software. Doing so would not only yield basic information about the frequency of spontaneous utterances bearing on aesthetic or embodiment themes, but also how that information is used in a dynamic creative problem solving context. For instance, do some themes emerge mainly in groping toward a first conceptualization of problem or a prospective solution, as a means of diagnosing or characterizing bugs or suboptimal solutions, of dynamically evaluating the performance of code (either positively or negatively), of formatting or otherwise packaging code in a way that is useful to other programmers (e.g., in collaborations or in free and open source environments), or in other ways?

It may be especially interesting to examine the role of embodiment via analysis of programmers' deployment of metaphor throughout the various tasks of software development. To the extent that programmers utilize metaphors in their creative problem solving, to what extent is there variation in the kinds of metaphors used? Does the notion of embodiment as a way of grounding aesthetics and metaphor presuppose a relatively unitary basis for metaphors, and thus a relatively unitary, constrained, or homogeneous basis for metaphoric reasoning, or is there scope for wider variation across individuals, across tasks, across metaphoric frameworks? Are there meaningful relationships between the founding metaphors of a programming language, the metaphors programmers use to frame and solve problems in that language? Milner's assertion about the power of object-oriented language to shape thought is echoed in the 'folk wisdom' that undergirds much of programmers' banter about language choice. For instance, one account of the rivalry between Perl adherents and Python adherents includes the characterizations, 'Perlites are chaotic/good trickster archetypes . . . [while] Pythonistas are peaceful, have-their-glasses-on-a-little-string types, like hobbits or the Dutch' (NTK 2004). While matching software development approaches with Dungeons and Dragons character types is likely a playful oversimplification, there may nonetheless be seeds here of software development methodologies which intentionally exploit the embodied dimensions of the code being produced, harnessing aesthetic judgment and other non-propositional forms of meaning and experience towards the production of code that is more fully habitable.

Of course this trajectory of study may also reveal new dimensions of embodiment, aesthetics, and metaphor, if we focus on these, rather than code itself, as our primary domains of inquiry. The flawed duality of the notion of disembodied cognition birthed with the modern computing machine has much to do with computer science's founding confusion in which, as Mark Poster describes it, 'the scientist projects intelligent subjectivity onto the computer and the computer then becomes the criterion by which to define intelligence, judge the scientist, outline the essence of humanity' (Poster, 1990: 148). While the focus of the 'digital humanities' on using computers as analytic/diagnostic tools for such aesthetic objects as may be reduced to digitized data, we propose the use of code as a vehicle by which we may decouple the machine from cognition, thereby coming to a deeper understanding of the phenomena which undergird meaning.

Endnotes

This work is funded in part by the US National Science Foundation, project number 0855861. The material presented here does not necessarily reflect the views of the National Science Foundation.

[1] In all likelihood, the phenomenon being described is what Csikszentmihalyi (1991) terms 'flow,' an experience with a number of characteristics including, sometimes, lack of bodily awareness and a suspension of the sense of the passage of time. Programmers' accounts of flow-like states tend to emphasize the locus of the experience as mental, with their bodies, sense perceptions, and so on, elsewhere or unavailable.

[2] See, for example, Grier (2007), on the history of human 'computers' (i.e., persons employed to manually perform numerical calculations with pencil and paper), which long pre-dates the modern history of electro-mechanical computers.

[3] Functional languages emphasize the idea of computation as the evaluation of mathematical functions, rather than the execution of a sequence of operations. Functional programs rarely include reference to underlying machinic structures.

[4] Even the notion of algorithm itself is grounded bodily: 'The very idea of an algorithmic process of calculation involves a starting point, a process that may or may not iterative, and a well-defined completion' (Lakoff and Nuñez, 2000: 37), which closely resembles the 'programs' carried out by our neural motor-control structures.

References

Arnheim, R. (1988) *The Power of the Center*. Berkeley: University of California Press.

Biancuzzi, F., & Warden, S. (2009) *Masterminds of Programming.* Sebastopol, CA: O'Reilly Media.

Boroditsky, L. & Ramscar, M. (2002) 'The Roles of Body and Mind in Abstract Thought', *Psychological Science* 13: 185-189.

Bourdieu, P. (1977) *Outline of a Theory of Practice.* Cambridge: Cambridge University Press.

Chandrasekhar, S. (1990) *Truth and Beauty: Aesthetics and Motivations in Science*. Chicago: University of Chicago Press.

Couger, J. D. *et al.* (1991) 'Using a Bottom-Up Approach to Creativity Improvement in Information Systems Development', *Journal of Systems Management* 42: 23-36.

Curtin, D. W. (ed.) (1982) *The Aesthetic Dimension of Science: 1980 Nobel Conference*. New York: Philosophical Library.

Csikszentmihalyi, M. (1990) *Flow: The Psychology of Optimal Experience*. New York: Harper and Row.

Dybvig, R. K. (2007) 'Syntax Abstraction: The syntax-case Expander', in *Beautiful Code,* (eds) Oram, A. & Wilson, G. Sebastopol: O'Reilly Media.

Ericsson, K. A. & Simon, H. A. (1984) *Protocol Analysis: Verbal Reports as Data*. Cambridge: MIT Press.

Feist, G. J. (2006) *The Psychology of Science and the Origins of the Scientific Mind.* New Haven: Yale University Press.

Gardner, H. E. (1987) *The Mind's New Science: A History of the Cognitive Revolution*. New York: Basic Books.

Glass, R. L. (2006) *Software Creativity 2.0*. Atlanta: Developer Books.

Gibbs, R. W. (2005) *Embodiment and Cognitive Science*. New York: Cambridge University Press.
Grier, D.A. (2007) *When Computers Were Human*. Princeton: Princeton University Press.

Hadamard, J. (1954) *The Psychology of Invention in the Mathematical Field*. New York: Dover.

Heisenberg, W. (1974) 'The Meaning of Beauty in the Exact Sciences', in *Across the Frontiers*, (ed.) R. Nanda. New York: Harper and Row.

Johnson, M. (2007) *The Meaning of the Body: Aesthetics of Human Understanding*. Chicago: The University of Chicago Press.

Kapor, M. (2002) http://www.longbets.org/1.

Kozbelt, A. (in press) 'Neuroaesthetics: Where Things Stand Now', *The Evolutionary Review*.

Kozbelt, A. *et al*. (2010) 'Beautiful Software: Characterizing Aesthetic Judgment Criteria of Code Among Expert and Novice Computer Programmers', given at the 2010 Biannual Meeting of the International Association of Empirical Aesthetics, Dresden, Germany.

Lakoff, G., & Johnson, M. (1999) *Philosophy in the Flesh: The Embodied Mind and its Challenge to Western Thought.* New York: Basic Books.

Lammers, S. (1986/2006) *Programmers at Work*. Redmond: Microsoft Press.

Leder, H. *et al*. (2004) 'A Model of Aesthetic Appreciation and Aesthetic Judgments', *British Journal of Psychology* 95: 489-508.

Levinson, J. (ed.) (2003) *The Oxford Handbook of Aesthetics.* New York: Oxford University Press. Locher, P., & Nagy, E. (1996) 'Vision Spontaneously Establishes the Percept of Pictorial Balance', *Empirical Studies of the Arts* 14: 17-31.

Maiden, N., Gizikis, A., & Robertson, S. (2004) 'Provoking Creativity: Imagine What your Requirements Could be Like', *IEEE Software* 21: 68-75.

Markman, A. & Brendl, C.M. (2005) 'Constraining Theories of Embodied Cognition', *Psychological Science* 16: 6-10.

Mauss, M. (1979) *Sociology and Psychology*. London: Routledge & Kegan Paul.

McAllister, J. (1996) *Beauty and Revolution in Science*. Ithaca: Cornell University Press.

Mohr, M. (1989) 'Programmed Esthetics', in *Esthetics, contemporary revised edition*, (ed.) R. Kostelanetz. Amherst, NY: Prometheus Books.

Molzberger, P. (1983) 'Aesthetics and Programming', *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* 247–50. Boston.

Niedenthal, P. *et al*. (2005) 'Embodiment in Attitudes, Social Perception, and Emotion', *Personality and Social Psychology Review* 9: 184-211.

Oram, A. & Wilson, G. (eds) (2007) *Beautiful Code*. Sebastopol: O'Reilly Media.

Otte, W.R. & Schmidt, D. C. (2007) 'Labor-Saving Architecture: An Object-Oriented Framework for Networked Software', in *Beautiful Code,* (eds) Oram, A. & Wilson, G. Sebastopol: O'Reilly Media.

Perkins, D. N. (1981) *The Mind's Best Work*. Cambridge: Harvard University Press.

Poster, M. (1990) *The Mode of Information*. Chicago: The University of Chicago Press.

Ramón y Cajal, S. (1937/1989) *Recollections of My Life*. (Translated by E. H. Craigie and J. Canto). Cambridge, MA: MIT Press.

Root-Bernstein, R. (2002) 'Aesthetic Cognition', *International Studies in the Philosophy of Science* 16: 61-77.

Rosenberger, S. (1997) 'Elegance and Entropy', http://www.salon.com/technology/feature/1997/10/09/interview/index.html

Saito, Y. (2008) *Everyday Aesthetics.* New York: Oxford University Press.

Silvia, P. J. & Brown, M. E. (2007) 'Anger, Disgust, and the Negative Aesthetic Emotions: Expanding an Appraisal Model of Aesthetic Experience', *Psychology of Aesthetics, Creativity, and the Arts* 1: 100-106.

Simonton, D. K. (2004) *Creativity in Science: Chance, Logic, Genius, and Zeitgeist*. New York: Cambridge University Press.

Snow, C. P. (1960) *The Two Cultures.* Cambridge: Cambridge University Press.

Tauber, A. I. (ed.) (1996) *The Elusive Synthesis: Aesthetics and Science.* Dordrecht, Netherlands: Kluwer.

Ward, T. B. *et al.* (eds) (1997) *Creative Thought: An Investigation of Conceptual Structures and Processes*. Washington, DC: APA.

Warren, H. S. (2007) 'The Quest for an Accelerated Population Count', in *Beautiful Code,* (eds) Oram, A. & Wilson, G. Sebastopol: O'Reilly Media.

Wechsler, J. E. (ed.) (1977) *On Aesthetics in Science.* Cambridge: MIT Press.