The Aesthetics of Software Code: A Quantitative Exploration

Article *in* Psychology of Aesthetics Creativity and the Arts · February 2012

DOI: 10.1037/a0025426

CITATIONS
5

READS
343

4 authors:

Aaron Kozbelt
City University of New York - Brooklyn College
83 PUBLICATIONS **1,175** CITATIONS
SEE PROFILE

Scott David Dexter
City University of New York - Brooklyn College
35 PUBLICATIONS **426** CITATIONS
SEE PROFILE

Melissa Dolese
CUNY Graduate Center
6 PUBLICATIONS **138** CITATIONS
SEE PROFILE

Angelika Seidel
City University of New York - Brooklyn College
10 PUBLICATIONS **249** CITATIONS
SEE PROFILE

Psychology of Aesthetics, Creativity, and the Arts
2012, Vol. 6, No. 1, 57–65

1931-3896/11/$12.00 DOI: 10.1037/a0025426

The Aesthetics of Software Code: A Quantitative Exploration

Aaron Kozbelt, Scott Dexter, Melissa Dolese, and Angelika Seidel
Brooklyn College and the Graduate Center of the City University of New York

While research linking science and aesthetics continues to proliferate, in technical domains like software development, quantitative investigations of aesthetics are virtually nonexistent. As an initial exploration, we administered an online survey to 12 experts and 38 novices in programming, assessing the frequency, nature, time course, and judgment criteria of their aesthetic experience with software code. Both groups reported having aesthetic experiences with code, though somewhat less frequently and intensely than with other creative artifacts. Overall, judgments of "ugly" code were reported to be faster than those of "beautiful" code, which in turn were faster than those of "correct" code. Aesthetic considerations of code were generally rated as quite important, though not as important as functionality. Finally, aesthetic judgment criteria were highly correlated among experts and novices. Results suggest a quantitative approach to aesthetics in software code is a promising direction, with trans-domain implications for aesthetics and creativity.

Keywords: aesthetics, computer code, expertise, aesthetic judgment, aesthetic experience

Many laypersons see science and aesthetics as largely nonoverlapping realms, paradigmatic of Snow's (1960) 'two cultures,' with little potential for productive cross-talk. Even among recent comprehensive reviews of scientific creativity (e.g., Feist, 2006; Simonton, 2004), aesthetics are given hardly a fleeting mention—a bias seemingly validating the 'two cultures.' This disconnect is also evident in research on aesthetics per se, as scholars have historically focused mainly on fine arts domains like painting and music (e.g., Leder, Belke, Oeberst, & Augustin, 2004; Levinson, 2003).

However, there are reasons to believe meaningful links exist between science and aesthetics. A number of scholars have suggested that aesthetic concerns apply to scientific domains (see, e.g., Curtin, 1982; Tauber, 1996; Wechsler, 1977). Others have argued that similar cognitive mechanisms undergird creativity in both the sciences and the arts. Such mechanisms can take a variety of forms, ranging from the chance-intensive exploitation of conceptual combinations in a Darwinian framework (e.g., Simonton, 1988, 2004) to largely rational problem solving processes that are guided by domain-specific expertise (Weisberg, 2006). Aesthetics are obviously relevant to art; if common psychological mechanisms underlie creative problem solving in many domains, aesthetics should likewise be relevant to science. Finally, from more of a personality psychology perspective, Feist (1998) found that defining traits of highly creative scientists include affective elements, specifically openness to experience, which has an important aesthetic component. Such research suggests there are strong reasons to question the traditional implicit exclusion of aesthetics from the realm of science.

First-person accounts of eminent researchers have often emphasized the importance of aesthetics in doing science. For instance, Nobel Laureate physicists such as Dirac (see Hovis & Kragh, 1993), Heisenberg (1974) and Chandrasekhar (1979, 1990) have written on the significance of beauty in the endeavor of physics. In Chandrasekhar's (1979) survey article on beauty in science, he poses a question: "All of us are sensitive to Nature's beauty. It is not unreasonable that some aspects of this beauty are shared by the natural sciences. But one may ask the question as to the extent to which the quest for beauty is an aim in the pursuit of science" (p. 25). He goes on to quote many esteemed researchers, mainly mathematicians and physicists, on the pervasive role of aesthetics in their scientific practice, suggesting that aesthetic concerns may be a primary motivating factor for scientists' continued activity as well as a basis of theory choice. The pioneering neurologist Santiago Ramón y Cajal likewise emphasized that science can be aesthetically rewarding, writing that in neurology his "aesthetic instincts found full satisfaction at last" (Ramón y Cajal, 1937/1989, p. 363). Root-Bernstein (2002) concurred that great scientific work has often resulted from a drive to experience beauty and added a stronger assertion: 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.

Even so, while research linking science and aesthetics continues to proliferate, the focus of research and discussion tends to be

This article was published Online First October 3, 2011.

Aaron Kozbelt, Melissa Dolese, and Angelika Seidel, Department of Psychology, Brooklyn College and the Graduate Center of the City University of New York; Scott Dexter, Department of Computer and Information Sciences, Brooklyn College and the Graduate Center of the City University of New York.

This material is based upon work supported by the National Science Foundation's "CreativeIT" Program, Grant No. 0855861, awarded to the first two authors, and PSC-CUNY Grant No. 61485-00 39 to the second author. Thanks to Michael Henderson, who assisted in the development of the survey and data collection, and to three anonymous reviewers, for their input.

Correspondence concerning this article should be addressed to Aaron Kozbelt, Department of Psychology, Brooklyn College of the City University of New York, 2900 Bedford Avenue, Brooklyn, NY 11210-2889. E-mail: AaronK@brooklyn.cuny.edu

mainly in nonapplied, highly theoretical domains. In contrast, in more applied or technical domains like software development, investigations of aesthetics are rare, and studies with a quantitative emphasis are virtually nonexistent. This may be due to the fact that the activity of writing software is poorly understood outside the software community. To what extent might aesthetic concerns be relevant to more technological domains such as computer science or software development? A common lay belief is that programming takes place in highly structured environments, relying solely on formal languages and standard techniques, with little or no room for creativity. Yet practitioners have a rich history of discussion and debate of the extent to which computer science is best understood as a science, as engineering, as a craft, or as an art.

Nevertheless, there is little doubt that writing software has the potential for creativity. The two traditional criteria for creativity, novelty and value (Runco, 2007), can be readily applied to software development. Along these lines, Glass (2006) strove to demonstrate the creative nature of programming—though the only empirical studies cited were Couger, McIntyre, Higgins, and Snow (1991) and Maiden, Gizikis, and Robertson (2004), both focusing on the effectiveness of creativity training on individual developers. If programming can be construed as a genuinely creative activity, how do aesthetics figure into this process? Unsurprisingly, the belief that programming is a genuinely creative act is most often espoused by programmers themselves (e.g., Lammers, 1986/2006; Oram & Wilson, 2007). Such accounts make the case that aesthetic considerations of computer code have substantial pragmatic consequences for developers and users alike, since aesthetics and beauty are common themes. Interestingly, programmers' descriptions of the role of aesthetics in software—both for programming languages and software applications—are often close in character to fine arts domains. To cite just one example: Gary Kildall, who developed the operating system CP/M, argued that "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" (Lammers, 1986/2006, p. 64).

Such a statement, which could be easily multiplied, reinforces the need to better understand and integrate aesthetic considerations into the creative activity of technical domains. However, a collection of first-hand accounts does not necessarily reveal the extent to which aesthetic considerations are a *pervasive* aspect of scientific or technical thinking; perhaps they are just characteristic of the best practitioners at their best moments. Alternatively, aesthetic concerns may be relatively common in technical domains, at least among persons with the requisite expertise.

While aesthetic concerns appear in many published accounts of programming—for example, in books with such titles as *Beautiful Code* (Oram & Wilson, 2007), *Hackers and Painters* (Graham, 2004), or *The Art of Unix Programming* (Raymond, 2004)—the particulars of these concerns are not entirely uniform. Many considerations might best be categorized as 'stylistic,' as in *The Elements of Programming Style* (Kernighan & Plauger, 1974), intentionally modeled after Strunk and White's (1972) treatment of English writing style; these include the layout of the code on the page/screen, naming practices, how and where to provide comments on the code, and the use of programming idioms. Other considerations seem to inhere at the nexus of an algorithm and its expression in code; these include such factors as the cleverness or ingenuity of the algorithm itself, the efficiency of the algorithm when applied to the problem at hand, and the aptness of the algorithm's expression in code. Still other considerations borrow heavily from architecture: the large-scale design or structure of a body of code, or the ease with which code might be modified, extended, or adapted to some future purpose. Finally, some considerations are expressed in language which evokes notions of improvisation, embodiment, and comfort; programmers such as those featured in Oram and Wilson (2007) and Lammers (1986/2006) are wont to use terms like *balance*, *flow*, *natural*, and *flexible*, while Gabriel (1998) proposes *habitability* as a major design goal for software. These varied accounts point toward the importance of aesthetics but do not shed much light on the question of how (and whether) aesthetics guides the programmer in the process of creating or revising code.

Methodologically there are many ways to explore this set of issues. Since the aesthetics of software code is woefully understudied, in this brief report we use a survey instrument to explore some basic questions about its nature, frequency, and judgment criteria, using samples of experts and novices. We thus attempt to go beyond the first-person descriptive accounts that, to date, have represented the primary source of information on this topic; while such data provide some sense of programmers' subjective experience of code, they are not necessarily amenable to quantitative analysis.

We argue that a quantitative approach to these issues represents a promising direction for addressing a host of fundamental questions on the nature of aesthetic experience in programming. For instance, how commonly is software experienced in an aesthetic way, compared to, say, visual artworks? How important are aesthetic considerations to the creation and evaluation of software, compared to considerations of functionality? What is the time course of experiencing or evaluating "beautiful" versus "ugly" code? What are the aesthetic judgment criteria for code among expert programmers, versus novices? The aim of this study is to provide some initial answers to such questions, in order to lay the groundwork for future investigations.

Method

Participants

Participants were recruited by e-mails to personal contacts in the computing industry and academia (where we hoped to recruit highly experienced programmers) and by announcements in undergraduate computer science courses (where we hoped to recruit programmers with substantially less experience). Because of the technical nature of the domain, true novices with absolutely no experience in computer programming or software development were not appropriate. Usable data were provided by 38 "novices," who were mainly undergraduates majoring in computer science, *M (SD)* age and years of programming experience = 25.4 (6.1) and 3.9 (2.0), and 12 "expert" computer science professionals, *M (SD)* age and years of programming experience = 41.6 (13.8) and 20.3 (11.7). The experts included computer science professors and professionals in the software industry. The distinction between experts and novices was based on the so-called 10 year rule for expertise acquisition (e.g., Chase & Simon, 1973), whereby a

minimum of a decade of serious involvement in a domain is required before a person can perform or create at a world-class level. In our sample, persons with 10 or more years of experience were categorized as experts. Fortunately, the bimodal distribution of years of experience lent itself to a ready split along this criterion.

Materials and Procedure

Potential participants were directed to an online survey instrument, posted on *surveymonkey.com*. The survey (see Appendix) was structured in the following way: (a) 19 questions on the frequency, intensity, nature, time course, and general importance of aesthetics (vs. functionality) in programming; (b) 62 brief descriptive phrases, each of which participants rated on an 8-point scale, in terms of how well each description characterized beautiful or aesthetically pleasing code—these were adapted from accounts by expert programmers, culled primarily from Oram and Wilson (2007); and (c) six demographic questions, assessing each participant's age, gender, and programming experience—length of time, intensity, context(s), and motivation(s). At the end, participants were also free to provide open-ended comments about the survey and about beautiful (and ugly) code in general. Data were collected during the Fall, 2009 semester. (Note: throughout the results, degrees of freedom occasionally differ due to missing data.)

Results

Frequency of Aesthetic Experience of Code Versus Other Creative Artifacts

The reported frequency of participants having *any* aesthetic experience with code was high: 100% of experts and 84% of novices, compared to 100% of experts and 92% of novices for creative artifacts in general—consistent with anecdotal reports that computer code can be experienced in an aesthetic way (Lammers, 1986/2006; Oram & Wilson, 2007). However, while most participants reported having had some aesthetic experience of code, such experiences were not overwhelmingly frequent, as can be seen in Table 1. The effect was somewhat modulated by programming experience: novices reported reliably less frequent aesthetic experience with code than with creative artifacts in general, Wilcoxon's $z = -2.76$, $p = .006$; experts did not show a statistically reliable difference. (Wilcoxon's test was used due to the clearly noninterval scale of answer options on the survey.) However, aesthetic experiences with code were not uncommon; indeed, some 16% of both samples reported having aesthetic experiences with code at least several times a month.

Emotional Strength and General Nature of Aesthetic Experience of Code Versus Other Creative Artifacts

In terms of the reported emotional strength of code-related aesthetic experiences, measured on a Likert scale ranging from 1 (*much weaker*) to 6 (*much stronger*), no reliable difference between experts and novices was found, $t(47) = 0.28$, *ns*, partial $\eta^2 = .002$, *M* (*SD*) ratings = 3.00 (0.85) and 2.89 (1.22) for experts and novices, respectively. In terms of the basic nature of code-related aesthetic experiences, measured on a Likert scale ranging from 1 (*very dissimilar*) to 6 (*very similar*), there was also no reliable group difference, $t(47) = 1.42$, $p = .16$, partial $\eta^2 = .041$, *M* (*SD*) ratings = 3.58 (1.16) and 2.92 (1.48) for experts and novices, respectively. Thus, for both groups, aesthetic experiences with code were somewhat weaker emotionally than other aesthetic experiences, but were only somewhat dissimilar in their basic nature, compared to participants' aesthetic experiences with other creative artifacts.

Self-Reported Importance of Aesthetics Versus Functionality, for Self Versus Other

Groups were also compared on the self-reported importance of aesthetic and functionality considerations, both in writing their own code and in evaluating code written by other programmers. In each case, importance was measured on a Likert scale ranging from 1 (*very unimportant*) to 6 (*very important*). Data were analyzed by a 2 (expert vs. novice) × 2 (aesthetic vs. functionality) × 2 (self vs. other) mixed-model ANOVA. Results included an overall main effect of group (experts vs. novices), $F(1, 44) = 4.42$, $p = .041$, partial $\eta^2 = .091$, with experts providing higher importance ratings, *M* (*SE*) ratings = 5.30 (0.30) and 4.59 (0.16), for experts and novices, respectively. Results also showed a main effect of functionality versus aesthetics, $F(1, 44) = 20.47$, $p < .001$, partial $\eta^2 = .317$, *M* (*SE*) ratings = 5.30 (0.30) and 4.59 (0.16), for functionality and aesthetic considerations, respectively, and a main effect of self versus other, $F(1, 44) = 11.26$, $p = .002$, partial $\eta^2 = .204$, *M* (*SE*) ratings = 5.09 (0.19) and 4.80 (0.16), for self and other, respectively.

Table 1
Frequencies of Self-Reported Aesthetic Reactions to Creative Artifacts (in General) Versus Computer Code, as Proportions of the Total Number of Participants in Each Sample

	Aesthetic reaction frequency: Creative artifacts		Aesthetic reaction frequency: Computer code	
Frequency	Experts	Novices	Experts	Novices
Never	.00	.08	.00	.16
Only Once or Twice Ever	.00	.18	.17	.34
Several Times Per Year	.50	.42	.67	.34
Several Times Per Month	.33	.21	.08	.16
Just About Every Day	.17	.11	.08	.00

Note. Proportions in each column may not sum to 1.00 due to rounding.

Among the possible interactions, the strongest was the interaction between group and self versus other, $F(1, 44) = 5.12$, $p = .029$, partial $\eta^2 = .104$, M (SE) ratings = 5.55 (0.33) and 5.05 (0.29) for self and other among experts, and 4.64 (0.17) and 4.54 (0.15) for self and other among novices. A marginal interaction was also found between aesthetic versus functionality considerations and self versus other, $F(1, 44) = 3.74$, $p = .060$, partial $\eta^2 = .078$, M (SE) ratings = 4.71 (0.21) and 5.48 (0.24) for aesthetic and functionality for self, and 4.16 (0.19) and 5.43 (0.22) for aesthetic and functionality for others.

Separate follow-up analyses of the aesthetic and functionality measures were also conducted, as 2 (group) × 2 (self vs. other) mixed-model ANOVAs. For aesthetics, two reliable main effects were found, echoing those reported above, but no interaction. For group, $F(1, 45) = 6.83$, $p < .001$, partial $\eta^2 = .132$, with experts providing higher ratings, M (SE) = 4.86 (0.32) and 3.92 (0.18), for experts and novices, respectively. For self versus other, $F(1, 45) = 12.81$, $p = .001$, partial $\eta^2 = .222$, with self receiving higher ratings, M (SE) = 4.65 (0.20) and 4.13 (0.19), for self and other, respectively. In contrast, no reliable effects emerged in isomorphic analyses of the importance of functionality, likely due to a ceiling effect, grand M (SE) = 5.44 (0.21); functionality appears simply too important to ignore in any programming context. However, in the absolute terms of the survey choices, aesthetic considerations were still judged to be quite important to programming. This was especially true among experts and applied more to programmers' own code, compared to that of others.

Time Course of Judgments

Our results also inform the reported time course of judgments of code, in terms of being beautiful versus ugly and correct versus incorrect ("correct" is the technical term for a program solving the problem it is intended to solve). As can be seen in Table 2, the distributions of reported time courses for experts and novices were similar for most of the measures, and nonparametric comparisons between groups on each measure (using the Mann–Whitney U test) were not statistically reliable. The only noteworthy trend was that for judgments of the beauty of code, novices showed a more or less bell-shaped distribution, while experts evinced a bimodal pattern, with almost half the sample reporting making beauty judgments in under a minute, and others reporting taking notably longer. This suggests that different experts focus on different aesthetic aspects of code, and/or have different understandings of the aesthetic nature of code—an issue potentially to pursue in future research.

In comparing the reported relative time course of judgments of beautiful versus ugly and correct versus incorrect code, more differences emerged. Specifically, each group indicated that judgments of ugly code were typically made faster than of beautiful code: for experts, Wilcoxon's $z = -2.70$, $p = .007$, for novices, Wilcoxon's $z = -4.20$, $p < .001$. In addition, novices (but not experts) claimed to make faster judgments of incorrect code, compared to correct code, Wilcoxon's $z = -2.25$, $p = .025$.

Interestingly, each group also indicated that judgments of beautiful code were typically made faster than of correct code: for experts, Wilcoxon's $z = -2.21$, $p = .027$, for novices, Wilcoxon's $z = -3.27$, $p = .001$. When asked whether it was easier to write code that was beautiful or correct, seven of the 12 experts and 26 of the 38 novices responded correct, a significant overall majority $\chi^2(1) = 5.12$, $p = .024$.

Aesthetic Judgment Criteria

Participants rated the extent to which 62 aesthetic judgment criteria characterized beautiful code, each on an 8-point scale. Across the criteria, average ratings of experts versus novices showed high agreement, $r(60) = .84$, $p < .001$. Equally weighting responses from both groups, the most strongly endorsed criteria included 'having comments that clearly explain nonobvious aspects of the program,' 'elegance,' 'using consistent indentation,' 'efficiency,' and 'being easy to modify/extend.' The only reliable group differences were found on the following items: 'is short,' 'is compact,' 'applies to large and small applications', and 'implements a small but important set of features from among a wide set of options' ($p = .005$, .004, .014, .015, respectively, for comparisons on these items). In each case, novices provided higher endorsements than experts. However, these differences should probably not be overstated, given the large number of t tests conducted and many statistically reliable correlations among individual items.

Discussion

This exploratory investigation sought to understand something of the nature of aesthetics in computer programming and software code among experts and novices, in a quantitative way. Our

Table 2
Self-Reported Time Courses of Judgments of Computer Code as Beautiful, Ugly, Correct, and Incorrect, as Proportions of the Total Number of Participants in Each Sample

	Beautiful		Ugly		Correct		Incorrect	
Time	Experts	Novices	Experts	Novices	Experts	Novices	Experts	Novices
Virtually instantaneously	.25	.16	.58	.37	.00	.03	.00	.03
<1 min.	.17	.11	.17	.32	.00	.03	.08	.05
1–2 min.	.08	.24	.25	.18	.08	.11	.00	.26
2–5 min.	.17	.24	.00	.05	.50	.29	.42	.32
5–30 min.	.33	.16	.00	.05	.25	.37	.33	.24
> 30 min.	.00	.11	.00	.03	.17	.18	.17	.11

Note. Proportions in each column may not sum to 1.00 due to rounding.

approach differs from the qualitative, descriptive accounts characteristic of previous writings on this topic (e.g., Lammers, 1986/2006; Oram & Wilson, 2007). We found that both experts and novices reported having aesthetic experiences with code, though somewhat less frequently and intensely than with other creative artifacts. Similarly, both groups rated aesthetic considerations of code as quite important, though not as important as functionality. Overall, judgments of ugly code were reported to be faster than those of beautiful code, which in turn were faster than those of correct code. Finally, the aesthetic judgment criteria of experts and novices were highly correlated. Methodologically, our findings also underscore the utility of a quantitative approach to understanding aesthetics in technical domains like software development, just as in fine arts domains (e.g., Hekkert & van Wieringen, 1996; Kozbelt, 2004; Kozbelt & Serafin, 2009).

Although our conclusions are necessarily somewhat tentative, they can be readily linked to more general scholarly discussions of creativity and aesthetics. Most broadly, our results reinforce the notion that aesthetic considerations are important in domains outside the fine arts. Although the frequency, intensity, and importance of aesthetic experience and concerns did not appear to be absolutely paramount in our samples, the results suggest that aesthetics play a sufficiently prominent role in the experience of software code to warrant continued investigation of this topic. In passing, we note that among creative domains, software design is probably not unique in subordinating aesthetics to functionality; we would predict similar patterns in other domains where functionality is absolutely vital, such as aeronautical engineering or architecture. More richly unpacking the relation between aesthetics and functionality across a range of domains remains a challenge for future research.

Some of our other results are relatable to more specific issues in aesthetics and creativity. For instance, the rather interesting finding that beauty appears more quickly discernible than correctness suggests that aesthetic-laden evaluative processes may drive judgment and decision making about software code—potentially both as final products and works in progress (see also Perkins, 1981). In other words, it may simply be more efficient for programmers to rely on aesthetic intuitions about code than to laboriously understand its details, when writing or revising code. Along these lines, since 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. We likewise note that the concept of judgments of ugly code itself speaks to research on another understudied topic, that of negative aesthetic emotions (e.g., Silvia & Brown, 2007).

Together with these positive or promising points, some caveats are in order. Naturally, as the first quantitative study of the topic of aesthetics and software code, replication of our results would increase confidence in their veracity. We likewise note that with our relatively small samples, some null results may have been observed due to limited power. Moreover, at least one finding, that of a strong similarity in the aesthetic judgment criteria of experts and novices, contrasts with the substantial expert-novice differences typically found in other domains (e.g., Kozbelt & Serafin, 2009). This may be attributable to the strong constraints of functionality in software development, or to the fact that our sample of novices had some training in programming, even if it was substantially less than that of the experts. More broadly, however, to detect meaningful expertise differences, future research may need to approach this issue in a more nuanced and contextualized way, using not just endorsements of general descriptors, but examining how different criteria are applied in the process of creating or modifying actual code artifacts. Ideally, quantitative and qualitative approaches would provide a convergent sense of the role and nature of aesthetics in software development.

Along these lines, apart from providing an initial set of quantitative results, some additional qualitative aspects of our data, not reported here, suggest some specific themes for more detailed follow-up. One concerns the notion of aesthetic induction and functional beauty, whereby usefulness establishes criteria for appraising beauty, continually reshaping the aesthetic canons on which subsequent works are evaluated (McAllister, 1996). Another such theme is that beauty appears to be more than just a way to describe objects or experiences; it also encompasses *embodied emotions*. The language used to describe beautiful objects is remarkably similar across domains. Programmers often describe their best work as elegant, flexible, natural, and flowing—terms evoking bodily experience, which are also used to describe great works of art. The language used to describe art echoes that used to describe our physical movement through space: music, for instance, takes us on journeys (Johnson, 2007); our physical interactions with the world ground our ability to sense compositional balance and 'the power of the center' (e.g., Arnheim, 1988) of visual artworks. Similarly, writings on the aesthetics of computer-generated art (e.g., Mohr, 1989) propose that technology becomes an extension of oneself; thus, unsurprisingly, terms related to the body are used to describe the software code which builds the implemented programs: if they to become an extension of oneself they must be clear, understandable, symmetrical, and natural. Relating the issue of embodiment to software is another theme worthy of empirical pursuit.

In conclusion, this study suggests that aesthetics are quite relevant to the domain of software development and, moreover, are empirically tractable. Many of these initial results can be related to the existing fine arts aesthetics literature and contribute meaningful hypotheses to the trans-domain study of aesthetics and creativity.

References

Arnheim, R. (1988). *The power of the center.* Berkeley, CA: University of California Press, Ltd.

Chandrasekhar, S. (1979). Beauty and the quest for beauty in science. *Physics Today, 32,* 25–30. doi:10.1063/1.2995616

Chandrasekhar, S. (1990). *Truth and beauty: Aesthetics and motivations in science.* Chicago, IL: University of Chicago Press.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology, 4,* 55–81. doi:10.1016/0010-0285(73)90004-2

Couger, J. D., McIntyre, S. C., Higgins, L. F., & Snow, T. A. (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, NY: Philosophical Library.

Feist, G. J. (1998). A meta-analysis of the impact of personality on scientific and artistic creativity. *Personality and Social Psychological Review, 2,* 290–309. doi:10.1207/s15327957pspr0204_5

Feist, G. J. (2006). *The psychology of science and the origins of the scientific mind.* New Haven, CT: Yale University Press.

Gabriel, R. (1998). *Patterns of software: Tales from the software community.* New York, NY: Oxford University Press.

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

Graham, P. (2004). *Hackers and painters: Big ideas from the computer age.* Sebastopol, CA: O'Reilly Media.

Heisenberg, W. (1974). The meaning of beauty in the exact sciences. In R. Nanda (Ed.), *Across the frontiers* (pp. 166–183). New York, NY: Harper and Row.

Hekkert, P., & van Wieringen, P. C. W. (1996). Beauty in the eye of expert and nonexpert beholders: A study in the appraisal of art. *American Journal of Psychology, 109,* 389–407. doi:10.2307/1423013

Hovis, R. C., & Kragh, H. (1993). P. A. M. Dirac and the beauty of physics. *Scientific American, 268,* 104–109.

Johnson, M. (2007). *The meaning of the body: Aesthetics of human understanding.* Chicago, IL: University of Chicago Press.

Kernighan, B. W., & Plauger, P. J. (1974). *Elements of programming style.* New York, NY: McGraw-Hill.

Kozbelt, A. (2004). Originality and technical skill as components of artistic quality. *Empirical Studies of the Arts, 22,* 157–170. doi:10.2190/NDR5–G09N-X7RE-34H7

Kozbelt, A., & Serafin, J. (2009). Dynamic evaluation of high- and low-creativity drawings by artist and non-artist raters. *Creativity Research Journal, 21,* 349–360. doi:10.1080/10400410903297634

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

Leder, H., Belke, B., Oeberst, A., & Augustin, D. (2004). A model of aesthetic appreciation and aesthetic judgments. *British Journal of Psychology, 95,* 489–508. doi:10.1348/0007126042369811

Levinson, J. (Ed.). (2003). *The Oxford handbook of aesthetics.* New York, NY: Oxford University Press.

Maiden, N., Gizikis, A., & Robertson, S. (2004). Provoking creativity: Imagine what your requirements could be like. *IEEE Software, 21,* 68–75. doi:10.1109/MS.2004.1331305

McAllister, J. (1996). *Beauty and revolution in science.* Ithaca, NY: Cornell University Press.

Mohr, M. (1989). Programmed esthetics. In R. Kostelanetz (Ed.), *Esthetics, contemporary* (Rev. ed., pp. 154–156). Amherst, NY: Prometheus Books.

Oram, A., & Wilson, G. (Eds.). (2007). *Beautiful code.* Sebastopol, CA: O'Reilly Media.

Perkins, D. N. (1981). *The mind's best work.* Cambridge, MA: Harvard University Press.

Ramón y Cajal, S. (1937/1989). *Recollections of my life.* (E. H. Craigie & J. Canto, Trans.). Cambridge, MA: MIT Press.

Raymond, E. S. (2004). *The Art of Unix programming.* Reading, MA: Addison Wesley.

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

Runco, M. A. (2007). *Creativity: Theories and themes: Research, development, and practice.* New York, NY: Academic 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. doi:10.1037/1931-3896.1.2.100

Simonton, D. K. (1988). *Scientific genius.* Cambridge, England: Cambridge University Press.

Simonton, D. K. (2004). *Creativity in science: Chance, logic, genius, and zeitgeist.* New York, NY: Cambridge University Press.

Snow, C. P. (1960). *The two cultures.* Cambridge, UK: Cambridge University Press.

Strunk, W., & White, E. B. (1972). *The elements of style* (2nd ed.). New York, NY: Macmillan.

Tauber, A. I. (Ed.). (1996). *The elusive synthesis: Aesthetics and science.* Dordrecht, Netherlands: Kluwer.

Wechsler, J. E. (Ed.). (1977). *On aesthetics in science.* Cambridge, MA: MIT Press.

Weisberg, R. W. (2006). *Creativity: Understanding innovation in problem solving, science, invention, and the arts.* Hoboken, NJ: Wiley.

Appendix

Code Aesthetics Survey

Instructions

In this survey, we are interested in your attitudes toward "aesthetic" aspects of computer code. Obviously, code can be assessed in terms of its correctness—whether it solves the problem it was designed to solve. Here we are interested in something else—not just whether code is correct, but the extent to which it can be considered "beautiful"—perhaps in a way that a painting, a symphony, or a mathematical proof can be considered beautiful.

There are no right or wrong answers to the following questions. Just answer each question honestly and efficiently. The survey should take no more than about 10 to 15 minutes. Your data will be completely anonymous—please do not put your name on the survey. If you are interested in the results, please contact Scott Dexter at *SDexter@brooklyn.cuny.edu*. Thank you for participating!!

Part One

In this section, circle one response for each question.

Have you ever personally had a strong aesthetic reaction to a creative artifact (i.e., a feeling of the beauty of a work of art, piece of music, mathematical proof, etc.)?

Yes No

If yes, how often?

Only once or twice ever——Several times a year——Several times a month——Just about every day

Have you ever personally had a strong aesthetic reaction to computer code (i.e., a feeling of the beauty of code)?

Yes No

If yes, how often?

Only once or twice ever——Several times a year——Several times a month——Just about every day

In terms of *emotionality*, my computer code-related aesthetic experiences have been:

Much weaker——Weaker——Somewhat weaker——Somewhat stronger——Stronger——Much stronger

than other aesthetic experiences I've had.

Please briefly explain:

In terms of the *nature* of the aesthetic experience (how it feels), my computer code-related aesthetic experiences have been:

Very dissimilar——Dissimilar——Somewhat dissimilar——Somewhat similar——Similar——Very Similar

to other aesthetic experiences I've had.

Please briefly explain:

In general, the time it takes me to determine how *beautiful* a one-page piece of code is:

Virtually instantaneously——Less than 1 min——1 to 2 min——2 to 5 min——5 to 30 min——More Than 30 min.

In general, the time it takes me to determine how *ugly* a one-page piece of code is:

Virtually instantaneously——Less than 1 min——1 to 2 min——2 to 5 min——5 to 30 min——More Than 30 min.

In general, the time it takes me to determine the *correctness* (i.e., meets specifications) of a one-page piece of code is:

Virtually instantaneously——Less than 1 min——1 to 2 min——2 to 5 min——5 to 30 min——More than 30 min.

In general, the time it takes me to determine the *incorrectness* (i.e., doesn't meet specifications) of a one-page piece of code is:

Virtually instantaneously——Less than 1 min——1 to 2 min——2 to 5 min——5 to 30 min——More Than 30 min.

When I write my own code, *aesthetic* considerations are:

Very unimportant——Unimportant——Somewhat unimportant——Somewhat important——Important——Very important

(*Appendix continues*)

When I write my own code, *functionality* considerations are:

Very unimportant——Unimportant——Somewhat unimportant——Somewhat important——Important——Very important

When I evaluate someone else's code, *aesthetic* considerations are:

Very unimportant——Unimportant——Somewhat unimportant——Somewhat important——Important——Very important

When I evaluate someone else's code, *functionality* considerations are:

Very unimportant——Unimportant——Somewhat unimportant——Somewhat important——Important——Very important

Overall, is it generally easier to recognize code as *beautiful* or *ugly*? (Circle one.)

Overall, is it generally easier to recognize code as *correct* or *incorrect*? (Circle one.)

Overall, is it generally easier to write code that is *beautiful* or *correct*? (Circle one.)

Part Two

In this section, you will read several short descriptions that may or may not characterize "beautiful" or "aesthetically pleasing" computer code. Please indicate the extent to which you agree with each description by writing a number from "1" to "8" next to each description, where "1" = Disagree completely, and "8" = Agree completely. Remember, there are no right or wrong answers. Just answer honestly and efficiently.

Beautiful Code

_____ Is compact
_____ Is easy to write
_____ Is "user-friendly"
_____ Is efficient
_____ Is elegant
_____ Is robust
_____ Is easy to modify and/or extend
_____ Is short
_____ Is easily maintainable
_____ Is correct in an absolute sense
_____ Is trustworthy
_____ Is simple
_____ Is easy to understand
_____ Is never *too* innovative
_____ Is unambiguous
_____ Is lengthy
_____ Is more likely to be found in complex software than in simple software
_____ Is an abstract virtue that exists independent of its programmers' efforts
_____ Is structurally organized in clear functional units
_____ Is redundant, with the same code in multiple places in a program
_____ Is conservative, sticking to traditional idioms most programmers are familiar with
_____ Is written in short lines of code
_____ Is frugal about resources like memory
_____ Helps a programmer be productive
_____ Has no frills
_____ Has scalability that comes from code optimization
_____ Implements a small but important set of features from among a wide set of options
_____ Has formatting that is related to the functionality of the code
_____ Shows high originality
_____ Tends to lead to improvements in execution speed
_____ Makes it easy to see how to modify and/or extend the program
_____ Deals with special cases cleanly

(*Appendix continues*)

_____ Is no more complex than necessary
_____ Breaks down a problem along natural boundaries
_____ Helps a programmer be happy
_____ Suggests ways to improve the code
_____ Obviously has no mistakes, rather than merely having no obvious mistakes
_____ Has one idea per line of code
_____ Is like a beautiful car—you rarely need to open it up to look at its mechanics; you can just enjoy the ride
_____ Is as linear as possible, not requiring a reader to jump around in the code more often than necessary
_____ Meets requirements and expectations on a wide range of conditions
_____ Speaks for itself, not requiring copious documentation
_____ Uses consistent indentation
_____ Has nearly every character of the program do useful work
_____ Functions as expected, not only in some basic or handpicked cases, but in *all* cases
_____ Never forgets that it will be running on a computer, and that a computer has limitations
_____ Has a relatively high proportion of blank lines
_____ Stands the test of time
_____ Implements a very generalizable solution
_____ Brings clarity to complexity
_____ Handles general cases uniformly
_____ Has scalability that comes from using the right algorithm
_____ Allows one to easily tell if the code is doing what it's supposed to be doing
_____ Applies to large and small applications
_____ Separates the expression of the desired computation from low-level implementation details
_____ Contains no unnecessary information
_____ Lightens the workload of the programmer
_____ Can be modified with a minimum of fuss
_____ Does not force programmers to do something against their intentions
_____ Balances multiple, potentially competing, considerations
_____ Makes similar aspects of a program similar in appearance in the code
_____ Allows a reader to infer logic from the visual appearance of code as well as from the code itself
_____ Has comments that clearly explain non-obvious aspects of the program

Part Three

Please answer the following demographic questions.

Age:

Gender:

How many years have you done computer programming?

In that time, how intensively have you been involved with programming? (Circle one.)

Very unintensively——Unintensively——Somewhat unintensively——Somewhat intensively——Intensively——Very intensively

In what context(s)? (Circle as many as apply).

Personal——Academic——Commercial——Nonprofit——Military——Government——Other (specify)

Identify the top one or two reasons why you do computer programming. (Circle one or two.)

Enhance career prospects——Enjoy the intellectual challenge——Contribute to a large endeavor——Exercise creativity——Improve skills——Other (specify)

Part Four (Optional)

We would like to improve this survey for future use. If you have any comments or suggestions, please feel free to write them here. In particular, if you can think of ways of characterizing beautiful (or ugly) code that differs from the descriptions in Part Two, please let us know. Thanks!

Received August 17, 2010
Revision received June 24, 2011
Accepted June 27, 2011 ■