Category: Software & Systems Design

Aesthetics in Software Engineering

Bruce MacLennan
University of Tennessee, USA

INTRODUCTION

It is commonly supposed that software engineering is—and should be—focused on technical and scientific issues, such as correctness, efficiency, reliability, testability, and maintainability. Within this constellation of important technological concerns, it might seem that design aesthetics should hold a secondary, marginal role, and that aesthetic considerations might enter the design process, if at all, only after the bulk of the engineering is done. This article discusses the important role that aesthetics can play in engineering, and in particular in software engineering, and how it can contribute to achieving engineering objectives.

BACKGROUND

Certainly, aesthetic considerations have not been completely absent from software engineering. For example, the development of structured programming ideas was accompanied with conventions for the textual layout of programs, which aimed for conceptual clarity, but also aesthetic appeal. (Even the older programming techniques relied on the aesthetic layout of flowcharts.) Knuth (1992) has advocated a practice of literate programming in which composites of programs and their documentation are treated as "works of literature." Elegance has been discussed as a criterion of programming-language design since the 1960s (MacLennan, 1997, 1999). Furthermore, software engineers have tended to prefer elegant algorithms (Gelernter, 1998) where the criteria of elegance have been inherited primarily from mathematics and theoretical science, in which beauty is a widely acknowledged value (Curtin, 1982; Farmelo, 2002; Fischer, 1999; Heisenberg, 1975; King, 2006; McAllister, 1996; Tauber, 1997; Wechsler, 1988; Wickman, 2005). In these and similar cases, however, there has been little direct work on an aesthetic theory for software.

One exception to the relative lack of explicit work on software aesthetics is the research program in aesthetic computing pursued by, for example, Fishwick (2002) and Fishwick, Diehl, Lowgren, and Prophet (2003). He argues that traditional program representations (such as textual programs and graphs) do not engage our aesthetic senses, and that they are abstract and uninteresting. However, because software concepts are abstract, they do not have a natural sensuous representation, and so Fishwick argues that they should be represented metaphorically (e.g., by visual or sculptural metaphors), and that the "craft-worthy, artistic step" is the choice of metaphor and the expression of the algorithm in terms of it. This metaphorical work (not the textual program, which is secondary) is then the primary medium for the aesthetic expression and appreciation of software (Fishwick, 2002). Recent developments in aesthetic computing are collected in Fishwick (2006).

Software engineering is a new discipline, and so it does not have a long aesthetic tradition as do many of the other arts and engineering disciplines. Therefore, it is helpful to look at well-established disciplines that have significant similarities to software engineering. One of these is mathematics, in which we may include theoretical science, which has in common with software engineering the fact that formal considerations dominate material considerations (explained below). As is well known, aesthetic considerations are important in theoretical science and mathematics (Curtin, 1982; Farmelo, 2002; Fischer, 1999; King, 2006; McAllister, 1996; Tauber, 1997; Wechsler, 1988; Wickman, 2005), and we will find Heisenberg's (1975) remarks on beauty in the exact sciences to be informative.

Another source of useful analogies comes, perhaps surprisingly, from the structural engineering of towers and bridges, which has in common with software engineering the fact that teams engage in the economical and efficient design and construction of reliable, large, complex systems, the failure of which may be expensive and dangerous. We will appeal especially to Billington's (1985) insightful investigation of the role of aesthetics in structural engineering.

AESTHETICS, FORMALITY, AND SOFTWARE

Importance of Aesthetics

Billington (1985) identifies three criteria of good design—the three *E*s of efficiency, economy, and elegance—which he associates with three dimensions of the design process: the scientific, the social, and the symbolic (the three *S*s). All of these also apply to software engineering. Efficiency is primarily a scientific issue since it deals with the physical resources necessary to meet the project's requirements. In structural engineering, the fundamental requirement is safety, which is also important in software engineering, in which other

important issues are real-time response, dynamic stability, robustness, and accuracy, among others.

The second criterion, economy, which treats benefits and costs, is a social issue because it depends on a society's values, both monetary and other. Certainly, many economical considerations can be reduced to money, which is almost the common denominator of value in the modern world, but it is problematic, at very least, to treat some costs, such as human death and suffering, in financial terms. The economy of a design depends on many social factors, including the costs of materials, equipment, and other resources; the availability and cost of labor; governmental regulations; and so forth. As a consequence, the economic worth of a design is more difficult to evaluate and predict than its efficiency.

This brings us to elegance, the aesthetic criterion, and while it will probably be granted that beautiful designs are preferable for their own sakes (other things being equal), Billington (1985) gives compelling arguments for the engineering worthiness of elegant designs, which also apply to software engineering.

As will be explained, elegance is a means of conquering complexity. In terms of their numbers of components and their interactions, software systems are some of the most complicated systems that we design. Even programs as simple as text editors may have hundreds of thousands of lines of code. Furthermore, these components (the individual programming-language statements) interact with each other in complex ways so that the number of potential interactions to be considered rises to at least the square of the number of lines of code. In addition, software engineers are at a disadvantage compared to other engineers for the components of programs are abstract operations, whereas designers in other engineering disciplines can depend on their physical intuition to aid their understanding (Ferguson, 1992).

Software engineers have designed various analytical tools to help them conquer this complexity, but it is important to understand the inherent limitations of analysis. Analysis makes use of models (especially mathematical models) to understand some system of interest. These models are useful because they make simplifying assumptions that permit the models to be understood more easily than the modeled systems. An effective model simplifies matters that are irrelevant to its purpose so that our limited cognitive resources can be devoted to the relevant matters. Often, the simplifying assumptions are made consciously; for example, we may use a linear model for a nonlinear system if we have reason to believe that the nonlinear effects are irrelevant to our concerns. However, models also typically make tacit simplifications because certain factors are assumed to be irrelevant or, in some cases, because some factors have never been considered at all. For example, Billington (1985) observes that the Tacoma Narrows Bridge collapsed 4 months after it was opened because aerodynamic stability had not been considered a factor in bridge design (see also Ferguson, 1992). The problem is more severe for more complex systems because our cognitive capacities are strictly limited, therefore so is the completeness of the model; hence, the gap between the modeled factors and the system increases with increasing system complexity. The risk of intentionally or inadvertently omitting relevant factors increases with system complexity.

Billington (1985) observes that in structural engineering, elegance compensates for the limitations of analysis. This is because in an elegant design, the disposition and balance of the forces are manifest in the design's form, and therefore designs that look stable are in fact stable. More generally, designs that look good are good. For example, the Tacoma Narrows Bridge and several other bridges, which were designed on the basis of extensive mathematical analysis and computer models, collapsed because aerodynamic stability was not included in the analysis, whereas earlier bridges, whose designs were guided by aesthetic judgment, were aerodynamically stable. If we generalize from the specifics of structural engineering, we may say that elegant designs are manifestly correct, safe, efficient, and economical. But why should we expect any correlation between aesthetic values and engineering values?

Billington (1985) argues that in structural engineering, Louis Sullivan's architectural maxim "form follows function" is not as applicable as its converse, "function follows form." Sullivan's principle asserts that a building's function should strongly determine its form. In structural engineering, however, there are typically many ways to fulfill a particular function (such as bridging a river at a particular location) for "engineering design is surprisingly open-ended" (Ferguson, 1992, p. 23). Therefore, since there is wide choice of form, the engineer can choose designs that manifest a stable disposition of forces, that is, elegant designs. More abstractly, engineers who are guided by elegance choose to work in a region of the design space in which aesthetic values correspond to engineering values (that is, function follows form). Therefore they can rely on their aesthetic judgment, which is a highly integrative cognitive faculty, to guide the design process.

The same considerations apply even more so in software engineering, where there are typically very many possible software solutions to an application need. That is, software engineers can compensate for the limitations of analysis by limiting their attention to designs that are elegant, that is, manifestly correct, efficient, maintainable, and so forth. In this region of the design space, where aesthetic values correspond to engineering values—where good designs look good—designers can rely on their aesthetic judgment to guide them to good engineering solutions.

Elegance is associated with the symbolic dimension of a design because, in the appropriate region of the design space, aesthetic values symbolize engineering values. However, design aesthetics can also symbolize less tangible