Studying the Philosophy of Software: A Framework for Examining How Digital Design Affects the Arts

Lindsay Grace, *Miami University, Ohio, USA*

Abstract: *The author extends his previous publications on the observed effects of software design decisions on the creative process by offering a framework for the philosophical evaluation of software designs. The task, as described, involves the systematic decomposition of assumptions and intended use as they effect the creative process.*

This research combines the established multidisciplinary examinations of critical design, post modern philosophy and creative process theory to define an innovative means of decomposing the effects of digital technology on creative production. The research takes case studies in creative writing and digital image production to demonstrate how the philosophical study of software design illuminates developer implied paths to production. Just as the design of a city directs pedestrians and cars, the design of software directs its users toward specific ends. A structured analysis of these implied paths, informed by critical examination through the lens of a variety of humanities (e.g. philosophy, social sciences, et al.) can yield engaging observations about the ways problems are solved with software.

Keywords: Software Philosophy, Creative Process and Software, Digital Design, Software Studies

Introduction

Software is the result of a managed creative process. It is produced by examining problems and deriving solutions through prioritizing those problems. It is in this standard process that the concept of software philosophy is most clear. The questions are simple. What are those problems and how have the software authors proposed to solve them? The investigation of these problems and their address are software philosophy.

Formally, software philosophy rests nearest the software studies discipline researched by Lev Manovic and others at UCSD (2008). Where software study is interested in the study of a general software society and its uses (Fuller, 2008), software philosophy is interested in revealing philosophical underpinnings of the decisions in the design and expected use of software (Grace, 2009). In particular, software philosophy is chiefly focused on the pedagogic and polemic content of software. It aims to apply a kind of design anthropology to the study of software solutions.

The content in this paper is designed to be accessible to a wide audience of readers familiar with general software tools. In particular it assumes some experience with very basic production tools. Readers should be familiar with productivity suits like Microsoft Office and image manipulation tools like Adobe Photoshop. Industry standard terminology is also used. If the reader is not familiar with the aforementioned tools or terminology this text may be challenging.

It should also be made clear that while at the surface, research in the management information systems area may have ancillary relationships to software and creativity, this writing does not intersect with that research. The use of decision support systems, for example, largely focuses on the qualities of information provided to individuals through software, instead of the effects software design decisions have on the way creative pursuit is executed. For more information about decision support and software, consider Elam and Mead's, Can Software Influence Creativity(1990) or MacCrimmon and Wagner's Stimulating Ideas through Creative Software (1994).

This brief writing outlines the process of revealing and researching three concepts in software philosophy as they relate to creative process in the arts. The first is the examination of feedback loops to expose the ways in which software interfaces can direct the user's problem solving approach. The second provides the framework through which software misuse can demonstrate software philosophies and reveal needed spaces in software design. The third provides perspective on software and its relationship to creative flow. These topics are explored using a combination of humanities and computer science driven inquiry.

The Feedback Loop-Structuring the Way we Think

In human computer interaction, designers routinely discuss the feedback loop. The feedback loop is the circuit between the user and the software they use. As a design philosophy it defines a continuous interaction stream of input and output. The user inputs information through a keyboard, the computer accepts that input and outputs the result through the screen, and the user responds by providing more computer input, for example. In practice, editing a paper in a word processing application, involves typing and mouse clicks as input, reviewing that input as words on the screen, and correcting any misspells as a response to the computer's output.

Software applications follow a fairly standard and somewhat rigid feedback loop. In the case of creative applications, the user is presented with a previously constructed workspace which is to be filled with their creative output. This workspace may be the blank page of a word processing application, the empty canvas of image manipulation software or a barren scene in a 3D software application.

Because so much of software is analogical, the expectation is that these things represent their appropriate real world equivalents. Before the computer, stories were written on pages, so the assumption is that a page must be an appropriate metaphor for eliciting input from the user and reporting output. Yet, like all simulation, there are some things that are explicitly missing from this likeness of a page. The pages in Microsoft Word, for example, assume top down, left-right writing. This assumption can be changed in settings, but the model dominates the initiation of writing project in the application.

More interestingly, the organization of the interface of Microsoft Word reveals an ontology of primary and secondary concerns when writing. According to the visual hierarchy of the application implied by order and placement of interface elements, the primary concerns in writing are formatting. The first page of Microsoft Word's editing panel is focused on graphic design, not writing. Correcting spelling,

for example, is not a primary concern. It is buried in the section under review, which implies that spell checking is part of the review process. While this may be a common way to write, this is not the only way to write. It merely indicates the philosophy of writing process encouraged by the application.

If you follow the visual cues of the application, the feedback loop implies a writing process as follows:

1. Write
2. Style
3. Insert non-text content
4. Layout the page
5. Clean up the writing

Now consider the writing process for someone working on a draft on paper:

1. Write
2. Revise
3. Review
4. Insert Content
5. Style

These are two perfectly appropriate creative processes for the writing. In this basic case, Microsoft Word does not prevent either process, although it does facilitate one better than the other. Playing to its own strengths as a tool, it could be argued that Microsoft Word biases toward the visual presentation of a written document philosophically.

Now consider the case of non-linear writing, as often employed in interactive media. In this writing process, the writer must organize the writing on some basis other than time. In such cases, what rests at the top of a page does not always logically follow what rests at the bottom of the page. This is contrary to the design standard of most word processors, which extend as you type from top-to-bottom. To edit non-linear narratives, most writers retrofit applications to meet their needs. Common solutions include using spreadsheets instead of word-processors or managing and linking multiple documents across a user defined file system.

Since linear writing is the dominant narrative style, most word-processing programs begin with it as their basis. This is not a short falling of the application, but it does effect the way we address the challenge of writing non-linear stories. Besides setting up unnecessary challenges to the less traditional problem of non-linear writing, there is a secondary effect of validation. Inexperienced users are driven toward prescribed solutions instead of being encouraged to solve it their own way. Original solutions are at the heart of creativity, but the systems software designers construct around our creative tools do not encourage us toward them. Where a physical page lacks the rigor of a software program, it also lacks the limiting prescriptions. Where there is no help file, there is often only creative way finding.

Consider the feedback loop in a misspelled word. The user misspells a word, the word is immediately identified as incorrect by underlining it in red. Imagine the same physical world analogy, a word is

written on a piece of paper and a teacher underlines the word before the sentence is complete. Even if the word is simply not in the grader's vocabulary, it continues to loom as a potential mistake. In such a world, difference is not creative, it is wrong. Imagine if Shakespeare wrote using Microsoft Word. We might not have the *eye ball* with which we view our software, the *employers* to which our template résumés are sent, or any *hint* of the more than 1600 words coined by the poet (Admson, 2001). From this perspective, Microsoft Word is a lousy tool for writing poetry, but wonderful for writing conforming prose.

This is because software demands use. As a manufactured item, it abounds with designer lead expectation and practical tolerances. By analogy, a block of wood does not demand use, other than a few adaptations to its material properties. A block of wood allows its user to construct it with other pieces, deconstruct it and shape it as needed. A block of wood can be manipulated into sculpture or worked into a fine piece of furniture.

Software as it is currently designed is more akin to pre-manufactured, ready to assemble furniture. It has a specific use, a prescribed process under which it is intended to be implemented. There is little point in breaking that intention because the software simply won't hold up. Alternative creative process is often the force that either breaks the application, or sends the user away.

Software Misuse as Design

As software demands specific uses, its intention is integral to our contemporary creative processes. Specifications are provided in a variety of circles in which the software dictates the output. If word-processed documents are requested, the requester often provides a list of accepted application formats. A job seeker, for example, would be asked for a résumé in specific formats. Likewise, presentations, digital images, and others are all constructed by the software applications associated with them. Consider if the same were true of art? What if medium dictated output? All works on canvas are to result in paintings; all works in metal are to result in sculpture. Even more analogous to the ways software is both used and designed, we not only dictate the result by medium, we determine the dimensions, shape and audience for the artifact.

From an artistic perspective, audience is often dictated by the technology you choose. If you make something with Adobe Flash you have the opportunity to share your work over the Web, design something with Yo Yo Games Game Maker and your audience is immediately limited to the Windows operating system. This is similar to creating non-digital works that are predetermined for an audience of the blind or deaf, for example. These are not distinctions of interest; they are distinctions of technical capacity. Where someone who is blind may not have the technical capacity to see, someone who does not have the required operating system lacks the technical capacity to view your work.

How does this effect the creative process? It starts at the beginning. Philosophically many software applications ask their users to make decisions about audience before they begin. Most versions of Photoshop, for example, ask the user to specify the size of their document before they begin their work. The document canvas can be resized, but some factors determined at this point cannot be undone

easily. The per-pixel resolution of the document, for example is far less malleable. This is an early point of commitment driven not by the creative process of the user, but by the software designer's prescribed workflow.

Consider how many software application tutorials begin with an explanation of concepts and principles of the application. Before a user can model a 3D object in Maya, they must understand the difference between non uniform rational b-spline (also known as NURBS) based modeling and polygonal modeling. That is, before you can work with the tool you must subscribe to its understanding of the world of 3D modeling. This indoctrination occurs, despite the users' existing understanding of 3D space.

Now consider the earlier example of non-linear writing. Spreadsheets software, such as Microsoft Excel, is routinely used to manage lists of storytelling dialogue in non-linear narratives. In some respects this is an explicit misuse of the software application, as spreadsheet programs are designed for the management of numerical, not text data. In other ways it is not a misuse, as the software is designed to favor lists and indices. These attributes make it more convenient than a traditional word processor for non-linear writing.

One effective way to expose software philosophies is to misuse software. This is not an exercise in mashing keys and clicking the mouse wildly. It is instead, an activity in careful, directed misuse informed by alternate understandings of the problem. In the previous case, non-linear writing is realigned from a problem of formatting and word processing, to a problem of list-making and index matching.

Consider two alternate perspectives on the process of making a virtual 3D model. Maya, a leading application for this task, is an explicitly additive creative process. Its prescribed creative process begins with emptiness and fills it with mathematical projections and geometry to simulate 3D space. Attempting to apply a subtractive creative process to Maya reveals some substantial challenges in using the application. Yet, as a creative process, sculptors of physical 3D artifacts find themselves adding and subtracting. A subtractive virtual modeling tool does exist, Z-Brush. In its creative process, 3D modeling begins with a material which can be reduced into specific forms. A misuse of Maya, inverting its additive model, reveals the need for subtractive tool. It is not surprising that Maya pre-dates Z-Brush.

An easy approach to revealing these software philosophies begins at their first step. Many applications ask for document type before they revealed their workspace. If a user begins an application in the wrong document type, and then works within those constrains they will expose some software philosophies. It will demonstrate, for example, how Microsoft Word defines the difference between a formal letter and a résumé. It will also demonstrate the absence of a template for poetry.

Software Viscosity and Flow

As a rule for evaluating software philosophies it is useful to understand that the more resistance created by software, the more likely it is to transform the way the user evaluates problems. This is derived from a fundamental rule in usability design – do not ask the user to adapt to the design, adapt to the user. Yet, even the most basic applications are designed against a fundamental framework that requires the user to stay on a designated track. Most interfaces for example offer a specific set of ways to accomplish

tasks. These are studied in user task analysis, and are typically part of an intended hierarchical path between user and their objective. Yet, by defining this route, the user is forced in a specific path. That path may not be the path of least resistance. This is one of the reasons that using physical paper to prototype or other analog means are so effective (Snyder, 2003). In programming we have *work arounds* for problems, in sculpture the artist works within the character of the material.

Working with a medium is the ideal. It allows the creative developer-designer to remain viscous. The designer-developer can work with the medium, smoothly integrating their own objectives with the fundamental restrictions of the software environment. A piece of paper may not stand up on its own, but by working with the character of paper, you can fold it, make curves or otherwise adapt it to your needs. Paper does not require much training, and it does not require the user to learn a new way to solve problems. Instead, it is part of a path of least resistance.

Fully constructed software systems, on the other hand, are not like paper. They are more like a toy car or a blender. They are designed against specific uses and have much more form to them. As such, adapting them to your needs is much more of a challenge. Yes, a blender and a toy car can stand up on their own, but they do not bend well and they cannot be curved easily. Instead, if you wanted to build a tower of blenders you would be working around their original design to make the tower stand. The original design worked well for its basic needs (blending and pouring), but as your needs change, the blender does not change. Constructed applications are not necessarily the path of least resistance.

Viscosity refers to the character of a liquid. IT describes the liquids resistance to force. Really great ideas often come from some balance of objective and adaptability. A person with creative motivation is much like a liquid moving toward their objective. There are things on the ground that might slow them down, but the more adaptable they are, the more likely they are to meet their objectives, because non-viscous (or thin) materials maintain their objective and are not easily stopped by a few obstacles. In the world of Psychology, this creative character is formally introduced as flow (Csikszentmihalyi, 1997).

When creative people start working with constructed software environments, we are placing that creative viscosity in a kind of pre-constructed maze. Every time a software philosophy enforces a particular creative approach, what was not viscous, becomes more viscous. Each time our idea gets momementum, we must wind that idea through all the walls created by software design decisions. This is commonly referred to as the work around. The software may have been designed for a set of specific reasons, but we work around those design intentions to meet our needs. Each time that adjustment is made, creative flow is stifled, and the free flowing creativity thickens. Simple solutions become complicated by assumptions in software design and consequently change the character of the creative flow.

As a means to an end, It is not that we as users are misusing the software by its intention; it is merely that we are misusing its design. Much like philosophy which has been turned against itself, the user is exposing the short fallings in the design concept to offer new perspective.

Conclusion

It is hoped that this brief introduction to a few of the key issues in software philosophy was helpful to the reader. As a field of research, the topic is new and full of the potential of fledgling research areas. The author is engaged in ongoing work to demonstrate alternate software philosophies through the production of a variety of software applications. The most notable of these are a collection of computer games which respond to dominant software philosophies in entertainment software. The games, published under the name Critical Gameplay, have been displayed at a variety of selective academic and artistic international venues. While the author will continue to investigate this area, it is also hoped that other researchers will investigate apparent software philosophies in their respective fields. In particular, the writing arts and visual arts seem ripe for such examination.

References:

Admson, S. (2001). Reading Shakespeare's dramatic language: a guide. In S. Admson. London: Cengage Learning EMEA.

Csikszentmihalyi, M. (1997). *Creativity: Flow and the Psychology of Discovery and Invention.* Harper Perennial.

Elam, Joyce J., Mead, Melissa.(1990) Can Software Influence Creativity? Information Systems Research. 1: 1-22

Fuller, M. (2008). *Software Studies: A lexicon.* Boston: M.I.T. Press.

Grace, L. The Philosophies of Software, Handbook of Research on Computational Arts and Creative Informatics (2009). IGI-Global Press, Hershey, USA,

Manovic, L. (2008). *Software Takes Command.* San Diego: http://softwarestudies.com/softbook/manovich_softbook_11_20_2008.pdf.

MacCrimmon, K. R., & Wagner, C. (1997). Stimulating Ideas Through Creativity Software. *Management Science, Vol. 40, No. 11* , 1514-1532 .

Snyder, C. (2003). *Paper Protyping.* San Francisco: Morgan Kaufmann.