View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Digital Commons@Wayne State University

DIGITALCOMMONS
@WAYNESTATE

Criticism

Volume 53
Issue 3 *Open Source*

Article 7

2011

Open Process Software

James J. Brown Jr.
University of Wisconsin, Madison, brownjr@wisc.edu

Follow this and additional works at: http://digitalcommons.wayne.edu/criticism

Recommended Citation
Brown, James J. Jr. (2011) "Open Process Software," *Criticism*: Vol. 53: Iss. 3, Article 7.
Available at: http://digitalcommons.wayne.edu/criticism/vol53/iss3/7

OPEN PROCESS SOFTWARE

James J. Brown Jr.

Expressive Processing: Digital Fictions, Computer Games, and Software Studies by Noah Wardrip-Fruin. Software Studies Series. Cambridge, MA: MIT Press, 2009. Pp 480. $35.00 cloth.

In *Program or Be Programmed* (2010), Douglas Rushkoff argues that we've ceded questions of software to a high priesthood of programmers:

> Our enthusiasm for digital technology about which we have little understanding and over which we have little control leads us not toward greater agency, but toward less . . . [W]e have surrendered the unfolding of a new technological age to a small elite who have seized the capability on offer. But while Renaissance kings maintained their monopoly over the printing presses by force, today's elite is depending on little more than our own disinterest. We are too busy wading through our overflowing inboxes to consider how they got this way, and whether there's a better or less frantic way to stay informed and in touch.[1]

Although Rushkoff's text moves a bit too quickly through complicated terrain, his larger argument holds water.[2] Most users of technology have remained just that—users. We have taken little interest in building our own tools or, at the very least, understanding how our tools are constructed. Despite the goals of engineers and designers such as Alan Kay and Douglas

© 2011 by Wayne State University Press, Detroit, Michigan 48201-1309

Engelbart, who hoped to create a situation in which all users would have the tools and know-how to write code, the role of *user* and *designer* have remained separate.

Humanistic scholarship provides a possible opportunity for those of us interested in addressing this problem. Attuned to questions of language and expression, humanists have begun to examine some of the ins and outs of programming. Scholarship in various disciplines has begun to take up the questions of the digital in earnest. But regardless of recent pushes to promote the digital humanities, these efforts have not been as widespread as some might hope. One can imagine some ambitious programs. For instance, computer programming could be taught broadly at the K-12 level and could be integrated into higher education beyond computer science programs. But in the meantime, we might look to smaller, incremental steps, such as the development of critical tools for better understanding how software works.

Noah Wardrip-Fruin's *Expressive Processing* provides some of those tools. Wardrip-Fruin's notion of "expressive processing" evokes two ideas at once. First, the term suggests that software is a significant and unique expressive medium that calls for users to pay careful attention to how processes have been authored. Second, the term allows Wardrip-Fruin to discuss "what processes express through their designs and histories" (5). Software bears traces of its design history, and Wardrip-Fruin hopes that his work can give us ways to recover that history. In addition to this titular term, Wardrip-Fruin develops a number of other critical concepts in the interest of providing game designers, artists, writers, new media scholars, gamers, and users new ways of considering the inner workings of software. Balancing so many different audiences is a difficult task. Indeed, Wardrip-Fruin suggests that two books are contained within *Expressive Processing,* one that argues that we "pay more attention to the processes of digital media" and another that provides a historical account of digital fiction and game design strategies (18). The text succeeds in balancing these tasks and audiences by providing detailed explanations of the theoretical apparatus, by putting that apparatus to work, and examining numerous examples.

Expressive Processing also works through some of the key questions posed by those of us interested in a bigger tent for computer programming. While scholars of new media will no doubt find Wardrip-Fruin's discussion useful, one goal of the text is to reach beyond the relatively small conversations of software studies (an emerging strand of new media scholarship) and digital fictions. Within this broader project, we might locate a promising expansion of the various political projects of open source and free software. For while free and open source software certainly allow for "more eyeballs" and "shallow bugs," they do not necessarily account for the development of software literacies; that is, the code itself may be open, but the average user has a limited set of tools for understanding how that code operates. *Expressive Processing* suggests some ways in which we might begin to theorize *open process software*—software that exposes its inner logics not by opening its code but via elegant designs. These open process designs would enable users to begin to parse and interpret the various logics at work below the surface.

Wardrip-Fruin's aim is to help both the general public and new media scholars develop a critical lens for software studies, and he contributes to this project by developing a number of concepts, including "operational logics," "the *Eliza* effect," "the *Tale-Spin* effect," and "the *SimCity* effect." *Operational logics* are patterns in the interplay among data, process, surface, interaction, author, and audience. Creators of digital media author data (the content of a story or quest) and processes (the various ways in which that data is arranged and delivered to the audience). Audiences interact with digital media at a surface level, and that interaction can change the state of the software's data and process while also revealing important details about how data and process interact. Wardrip-Fruin's three effects provide various ways by which we might understand this relationship between internal operations and surface effects.

The *Eliza* effect, named after Joseph Weizenbaum's famous natural language processing system, describes a situation in which complexity at the surface of a digital system leads users to assume internal complexity. Although the term is not Wardrip-Fruin's, *Expressive Processing* provides a rethinking of the concept. In the mid-1960s, Weizenbaum created Eliza, a precursor to today's chatbots that enabled users to "talk" with a therapist, and users began to assume that the program was able to carry on a complex conversation. This illusion often collapsed as users attempted to sustain longer conversations with Eliza, and they soon discovered that the system was merely manipulating user-entered strings of text in rather simplistic ways. Nonetheless, the breakdowns that happen as users interact with Eliza provide a useful way of understanding the relationship between user experience and internal processes. Wardrip-Fruin suggests that the *Eliza* effect has led many designers of interactive fiction and games to avoid the problem altogether. That is, rather than run the risk of an *Eliza*-like breakdown, designers have begun to make it clear at the outset that they're not attempting to create artificial intelligence (AI) that approximates human behavior. Indeed, one of Wardrip-Fruin's key insights is that many designers and authors are not typically interested in creating accurate models of the world. Rather, their main goal is to create models with which users interact. The goal is not to recreate the "real" world but rather to create a world.

Wardrip-Fruin's second intervention is the *Tale-Spin* effect, named after James Meehan's attempt to write a metanovel. *Tale-Spin,* Meehan's story-generation machine, generated simplistic, nonsensical, confusing narratives, which has led most scholars in the humanities to dismiss the system. However, most computer science discussions of the software "tend to treat the system as worthy of serious engagement" (121). Wardrip-Fruin's analysis bridges this gap by bringing the expertise of a software designer to bear on the concerns of humanistic scholarship. He does this by explaining his *Tale-Spin* effect: "The Eliza effect creates a surface illusion of system complexity—which play (if allowed) dispels. The Tale-Spin effect, on the other hand, creates a surface illusion of system simplicity—which the available options for play (if any) can't alter" (146). Meehan's system spun nonsensical, simplistic stories, but the simple output of the system was by no means the result of simplistic design. *Tale-Spin* actually employed a fairly complex *planboxing* technique in an attempt to generate stories. As Wardrip-Fruin's painstaking analysis of *Tale-Spin*'s operational logics attests, this system has a great deal to tell us about the authoring of digital media as well as the relationship between user experience and internal processes. *Tale-Spin* may have failed to generate compelling stories, but it was a complex system. We can fully understand that complexity only if we look beyond surface effects.

If the *Eliza* effect and the *Tale-Spin* effect indicate the failures of a system to negotiate the relationship between surface effects and internal operations successfully, then the *SimCity* effect describes software that strikes a balance. In a game like *SimCity*, the player is encouraged to develop a deeper understanding of the system's internal processes. The game does not attempt to present an authentic model of artificial intelligence, and it is quite clear to players of *SimCity* that they are not interacting with humans. However, the game does succeed in clearly communicating its operational logics: "[T]he elements presented on the surface have analogues within the internal processes and data. Successful play requires understanding how initial expectation differs from system operation" (302). Thus, players build a model of the system through a process of trial-and-error, and *SimCity* avoids the pitfalls of games that attempt (and fail) to model human behavior accurately. Further, it provides an important model for those of us seeking systems that can educate nonprogrammers about how software works. As Wardrip-Fruin explains,

> [T]he example of *SimCity* is important to our culture precisely because it demonstrates a way of helping millions of people develop a type of understanding of complex software models. This understanding, again, is not detailed enough for reimplementation—but rather like the gardener's understanding of interacting plants, soil, weather, weeding, and so on. A gardener doesn't need to understand chemistry, and a *SimCity* player doesn't need to understand programming language code, yet both can come to grasp the elements and dynamics of complex systems through observation and interaction. (310)

In this case, gameplay offers users a glimpse into the models built by the *SimCity* designers. The game models what we might call an open process approach, one that signals a hopeful direction for those hoping to educate nonprogrammers about how software works.

While the open source and free software movements have meant that we have access to more code, it has not guaranteed a broad conversation about the cultural, ethical, and political effects of software design. Further, as Wardrip-Fruin argues, the examination of code does not tell the whole story:

> [I]f we think of software as like a simulated machine, interpreting the specific text of code is like studying the choice and properties of materials used for the parts of the mechanism. Studying processes, on the other hand, focuses on the design and operation of the parts of the mechanism. (164–65)

Understanding software means digging beyond surface effects. Surfaces can tell us about internal processes, and we should understand ways of making such connections. For Wardrip-Fruin, this is a question of software literacy and not just a scholarly endeavor. Further, we should have methods for digging beyond the surface so that something like the *Tale-Spin* effect can come into full relief.

Wardrip-Fruin's afterword is devoted to explaining the blog-based peer review of *Expressive Processing*. In addition to feedback from MIT Press reviewers, he solicited feedback from the readers of *Grand Text Auto,* an academic blog. This approach is a promising one for those interested in issues of open access publishing, and the afterword works through some of the promises and complications of a more open peer-review process. However, to my mind, the book's most important contribution comes in its insistence that scholars across various disciplines can and should begin developing ways for average users to think more critically about software. As Wardrip-Fruin explains, such a project is not one confined to academic circles:

> In our society we are surrounded by software—from everyday Google searches to the high stakes of Diebold voting machines. We need to be prepared to engage software critically, accustomed to interpreting descriptions of processes, able to understand common pitfalls, and aware of what observing software's output reveals and conceals about its inner workings. (422)

One does not need to write code to be software literate. While writing code is certainly the best way to gain a deep understanding of how software works, we might also consider paying closer attention to the environments in which we are called to work and play. And though Wardrip-Fruin concedes that understanding the operational logics of software may not make us all software designers, he suggests that games like *SimCity* can at the very least "produce a kind of feeling for the algorithm, for processes at work, for potentials and limits" (395). This feeling for the algorithm can happen if we begin to think beyond open source software and begin to consider what open process software looks like. The latter would call for design strategies that reveal underlying processes and for nonprogrammers who learn to pay careful attention to those processes. It would call for a sustained conversation about how software works and how it works us over.

James J. Brown Jr. is an assistant professor of English at the University of Wisconsin–Madison. His research focuses on digital media, rhetoric, and writing, and he teaches in the Digital Studies program.

NOTES

1. Douglas Rushkoff, *Program or Be Programmed: Ten Commands for a Digital Age* (New York: OR Books, 2010), 140–41.
2. Although Rushkoff is no doubt writing for a popular audience, his tendency toward punditry leads to some specious arguments. For instance, he argues that the difference between an analog record and a digital CD is that the record "is the artifact of a real event that happened in a particular

time and place," whereas a CD "is not a physical artifact but a symbolic representation" (46–47). While there are indeed important differences between vinyl recordings and digital ones, Rushkoff's argument ignores the materiality of digital data that a number of scholars, including Matthew Kirschenbaum, have gone to great pains to demonstrate.