Conceptual Computing and Digital Writing

For Postscript, edited by Andrea Andersson. Draft of 24 September 2014.

Nick Montfort

In 1952 computer scientist Christopher Strachey wrote a parodical love letter generator.[1] This system, the prototype of all computational conceptual writing – the almost completely secret prototype – was up and running not only before conceptual writing was formulated but even before conceptual art had arrived. The program predates the earliest work that is consistently identified as part of the (yet unnamed) conceptual art movement, Rauschenberg's *Erased De Kooning Drawing.* It was not created by someone who identified or was identified as a writer, or as an artist, and it seems to have been seen as more the server-room equivalent of a parlor game than as a part of the tradition of literary arts. Only recently have programmers and scholars provided versions of the generator that appear in an installation and Web contexts[2] and discussed in depth the literary aspects of the system.[3] All of this makes Strachey's program not only the first in its category but also quite typical of the scattered, marginal, often overlooked projects that have explored the computer's ability to write conceptually over the last sixty years.

The computer has certainly not been ignored by conceptual writers, but those who have discussed the role of computing have often pointed to its place in the writing and reading environment and looked to it as a tool to rip and burn words (uncreatively, of course) rather than to work and play with them. Christian Bök, for instance, identifies some of the important qualities of the networked computer as they pertain to innovation in poetry:

> Recent trends in technologies of communication (such as digitized sampling and networked exchange) have already begun to subvert the romantic bastions of "creativity" and "authorship," calling into question the propriety of copyright through strategies of plagiaristic appropriation, computerized replication, and programmatic collaboration. [4]

Bök identifies a powerful relationship between computing and contemporary writing, but his comment focuses on how the computer challenges existing, outmoded concepts and institutions. He does not celebrate the computer as a machine with a new type of "creativity" (or a poetics), but as a tool for duplicating texts (and, will be discussed later, collaboration). It is notable that Bök, in writing his univocalic masterwork *Eunoia*, did not use a computer to generate word lists; rather, he read through an unabridged dictionary, in codex form, seven times.

There are systems before the electronic computer that could create combinatorial texts, such as Raymond Lull's paper machine, the Lullian circle. This device was developed around 1275 and included in copies of Lull's *Ars Magna;* it was designed to generate all true propositions about God. As important as these predecessor machines are, the late 20th-century programs written for the general-

1 Strachey, Christopher, [*Loveletters.*] Mark I Autocode program for the Ferranti Mark I.

2 The exhibit first went up at ZKM: Link, David, *LoveLetters_1.0. MUC=Resurrection. A Memorial,* ZKM, Karlsruhe, Germany, April-August 2009. The Mark I emulator can be found at: Link, David, *Manchester and Ferranti Mark 1 Emulator,* http://www.alpha60.de/research/muc/

3 Wardrip-Fruin, Noah. "Digital Media Archaeology: Interpreting Computational Processes," *Media Archaeology: Approaches, Applications, and Implications,* eds. Erkki Huhtamo and Jussi Perikka (Berkeley: University of California Press: 2011) 302-322, <http://games.soe.ucsc.edu/sites/default/files/nwf-BC7-DigitalMediaArchaeology.pdf>

4 Bök, Christian, Interview at the Canadian Literature Symposium, Ottawa, Canada, by Brenda Dunn, 2008, <http://www.canlit-symposium.ca/past/2008/interviews/dunn-bok-final.pdf>.

purpose electronic computer were not a simple continuation of practices that had been undertaken with dice, with slips of paper in a hat, and with other mechanisms.[5] The discussion here focuses on electronic text generation from Strachey's system to the present.

Starting with Strachey's computer program and the context in which it arose allows for a discussion of computational conceptual writing that historicizes this thread of practice and acknowledges its innovative qualities and its mostly outsider status – even within the renegade practice of conceptual writing. This discussion also provides a typology along four axes, characterizing different sorts of systems according to whether they sample or enumerate; whether they use a static or dynamic supply of source texts (or textons); their level of complexity; and their use of text only or multiple media. This formalism is not meant to capture every property of these systems, but to show that systems to write computationally and conceptually have identifiable independent aspects, and that they are very poorly characterized by single terms, such as "multimedia," used alone.

First Dimension: Exhaustive vs. Sampled Texts

Any given computer text generator defines some set of texts, those texts that it can produce. An exhaustive generator, when it completes execution, will have produced all of them. A generator that samples, on the other hand, provides individual instances of text, choosing them from a distribution, a set in which each element is weighted with a probability.[6] The latter kind of generator is certainly more common, but computer writing systems of both sorts exist.

Consider a simple Python program that generates names that sound like (or in some cases actually are) art movements:

```
first = ['computer','conceptual','net','pop','real']
second = [' art',' ism',' school']
for a in first:
    for b in second:
        print a + b
```

The output of this program is a list of 15 phrases – every combination of the five "first parts" and the five "second parts." The list begins:

```
computer art
computerism
computer school
conceptual art
conceptualism
...
```

This program can be quickly typed into a Python interpreter; the reader is invited to do so. It is an exhaustive writing system. With slight modifications, it can become a sample-based generator:

```
from random import choice
first = ['computer','conceptual','net','pop','real']
```

5 For more on the connection between pre-digital and digital text-generating machines, see Trettien, Whitney, "Computers, Cut-Ups, & Combinatory Volvelles: An Archaeology of Text-Generating Mechanisms," masters thesis, MIT, May 2009, <http://www.whitneyannetrettien.com/thesis/>, which is itself a digital text-generating machine.

6 Because I believe it is most important to distinguish exhaustive from sampling generators, I am overlooking some further complexities: How input from the user influences the text produced, for instance, or how the generation of particular texts may be conditioned upon previous outcomes rather than being independent of them. The exhaustive/sampling distinction should help to explain the role of randomness in text generation, to describe how it is not essential to every type of generation, and to lay a foundation for further, deeper consideration of systems involving randomness.

```
second = [' art',' ism',' school']
print choice(first) + choice(second)
```

So changed, the program prints out one of the fifteen possible outputs at random each time it is run. Specifically, because "choice" samples uniformly at random,[7] each of the five beginnings and each of the three endings are selected equally often, and each of the fifteen outcomes is equiprobable. The program might produce "netism," for instance, but any of the other fourteen outcomes is equally likely.

Randomness is used in sampling a particular text, but randomness is not the quality that distinguishes a sampling generator from an exhaustive one. An exhaustive generator could shuffle its output texts randomly before printing them and it would still be exhaustive.

Sampling: Loveletters, Stochastic Texts, A House of Dust, about so many things

Loveletters produces a parody document, full of meaningless endearments, by generating an address, five sentences (each of which can be of two different forms), a salutation, and the signature "M. U. C." (Manchester University Computer). The result, for instance, can be:

MOPPET SWEETHEART

MY DEAR AMBITION EAGERLY IS WEDDED TO YOUR LUST. YOU ARE MY AVID FONDNESS. MY THIRST IS WEDDED TO YOUR FANCY. YOU ARE MY CURIOUS DEVOTION. MY TENDER ENTHUSIASM ARDENTLY SIGHS FOR YOUR SWEET FONDNESS.

YOURS ARDENTLY

M. U. C.

Although Strachey's fame does not rest on his achievements as a writer or artist, he did publish about this project in a literary and art review, *Encounter*, noting "the remarkable simplicity of the plan when compared with the diversity of letters that it produces." He also explained that the vocabulary was drawn from a book much favored by conceptual writer Vito Acconci: Roget's Thesaurus.[8] And, importantly, he noted how the point of this exercise was not to see how the program could be improved but to understand, through it, how simple "tricks can lead to quite unexpected and interesting results."[9]

Theo Lutz took a more canonical literary source – Kafka – for his 1959 program to generate bizarre propositions. His *Stochastic Texts,* generated on a Zuse Z22 computer, include such statements as (in English translation from the German) "A CASTLE IS FREE AND EVERY FARMER IS FAR." and "NO COUNT IS QUIET THEREFORE NOT EVERY CHURCH IS ANGRY." His system's operation, and its results, were consonant with Kafka's description of a formally valid social system in which the particular combinations were often meaningless. This system was also described and discussed in an art

7 To be precise, the computer is producing not randomness but "pseudorandomness." Early implementations of randomness on the computer were complex deterministic functions. Today, the quest to have computers produce true randomness – by whatever definition; there are many – continues. Much more on this point, and its relevance to aesthetic computing, can be found in the "Randomness" chapter of Montfort, Nick et al., *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*, The MIT Press: 2012. Computers today, whatever their defects, are tremendously better at randomization than are people. The distinction between pseudorandomness and randomness has subtle aesthetic implications, but does not significantly complicate any of the discussion here.

8 My point here is not to defend *Loveletters* as a legitimate art project, but to show that, even though it was developed in the context of an early computer lab and not apparently intended for art-world viewers or a broad group of literary readers, it has some connection to the art world and even to later conceptual writing practices.

9 Strachey, Christopher, "The 'Thinking' Machine," *Encounter* (1954) 27.

journal, Max Bense's *Augenblick.*[10]

A House of Dust[11] is another significant early combinatorial text generator. Alison Knowles and James Tenney collaborated on this program, which produced stanzas using one simple template:

A HOUSE OF [material]
[location]
USING [light]
INHABITED BY [people]

For instance, this template and the lists of terms could (and did) generate:

A HOUSE OF DUST
ON OPEN GROUND
USING NATURAL LIGHT
INHABITED BY FRIENDS AND ENEMIES

The system is significant for its use within an art practice and for being situated within an art movement. The stanza shown here inspired the construction of an architectural and sound work by Knowles beginning in 1968. The work was not only part of an artistic process; its output was also presented to readers. A chapbook-like publication of stanzas presented them on 20 sheets of fanfold computer paper, foregrounding the material nature of computer output.

The stanzas generated by *A House of Dust* are compelling because they are evocative and activate the imagination – one can hardly help but fill in details about how the house is constructed, how exactly it is situated, and who resides there – but also because they balance a diversity of terms with pleasing repetitions that cause the stanzas to cohere and connect. Funkhouser writes that the system's "flexibility or variation spares the cumulative poem from being monotonous,"[12] which is certainly correct, but it is also important to note how the regularity of form, enforced by the single template, and the occasional recurrence of phrases are both also essential to the success of *A House of Dust.*

In 1974, Canadian artist Norman White created what might be now called an example of "physical computing," although it did not use a general-purpose computer. His *Four-Letter Word Generator* is a custom circuit with four neon alphanumeric readout tubes. It was engineered to produce pronounceable words which were not all English dictionary words.[13] The system produces some of the most concise computer-generated texts, simpler, for instance, than Lutz's terse propositions. A recent project along somewhat similar lines, in that it is a custom electronic device to generate texts combinatorially, is Adam Parrish's *Autonomous Parapoetic Device,* created in 2008. Parrish's text generator uses a general-purpose, programmable microcontroller and a larger display of 20x4 characters.[14]

A combinatorial generator that is remarkable for its simplicity and effectiveness is Nannette Wylde's *about so many things,* one of her Electronic Flipbooks from 1998. The systems in this series were written in Macromedia Director and used a straightforward template with lists of text options. In "about so many things," the template is simply:

He [sentence completion]

---

10 Lutz, Theo, "Stochastic Texts," *Augenblik* 4:1 (1959) 3-9. English translation by Helen MacCormack, 2005, <http://www.stuttgarter-schule.de/lutz_schule_en.htm>.

11 Knowles, Alison and James Tenney, *A House of Dust* (Cologne: Verlag Gebruder Konig, 1968). The chapbook is the output of a Fortran program from 1967.

12 Funkhouser, C. T., *Prehistoric Digital Poetry: An Archaeology of Forms, 1959-1995* (Tuscaloosa: The University of Alabama Press, 2007) 61.

13 White, Norman, "Norm the Artist, Oughtist, and Ne'er-do-well," <http://www.normill.ca/artpage.html>.

14 Parrish, Adam, "Autonomous Parapoetic Device (APxD mkII)," Decontextualize.net, <http://www.decontextualize.com/projects/apxd/>.

She [sentence completion]

The READ_ME file for *about so many things* explains: “the activities are drawn from the same pool of possibilities. Any line of text could be applied to either subject. In essence, the work explores the release of societal constraints regarding gender roles.” The sort of texts that result are:

He is a successful entrepreneur
She has a secret

The sentence completions range from being almost gender-neutral to being of quite different valence when applied to people of different genders. They include: “likes chocolate,” “feels stressful,” “is a good parent,” “has a crush on the teacher,” “is wearing makeup,” “is a firefighter.” By simply assigning sentences at random to be about either “He” or “She,” *about so many things* produces interesting texts that expose the reader’s individual and cultural preconceptions related to gender.

All of these examples help to show how the development of text generators is distinct from the typical process of writing texts. A traditional writing practice involves specifying *sequences of words.* For a fiction writer, these will typically form sentences and paragraphs; for a poet, they will be lines and stanzas. The creator of a combinatorial text generator instead defines *a distribution over sequences of words*. The program defines a set of possible texts – as should be clear from the examples above – but also the way each text in the set is weighted. Since this is less evident from the programs discussed, consider the following BASIC program:

```
10 IF RND(1)>.5 PRINT "HEADS": GOTO 10
20 PRINT "TAILS": GOTO 10
```

In it, the words “HEADS” and “TAILS” are selected with equal probability – the selection is done uniformly at random. But if “.5” is changed to “.2”, what was initially a fair coin flip becomes biased. Each time, there will be an 80% chance that “HEADS” is printed. The set of options is the same in both cases, but the distribution is different because the weight on each of the two options changes.

The writer of a sample-based combinatorial text generator is not only defining every possible text that the system can generate; he or she is also assigning weight to each possible outcome. In many cases, a uniform distribution over a compelling set of texts works well. Some text generators of this sort are effective because certain outcomes are more rare and others more common, however.

Exhaustion: Permutation Poems

Byron Gysin’s permutation poems, created in collaboration with Ian Sommerville beginning in 1959, are the output of exhaustive systems.[15] They consist of phrases which in most cases have been rearranged by computer. The first one, for instance, lists all 120 permutations of “I AM THAT I AM,” starting with the lines:

I AM THAT I AM
AM I THAT I AM
I THAT AM I AM
THAT I AM I AM
AM THAT I I AM
...

Because there are only three unique words in the original five-word phrase, the full poem contains

15 Gysin, Brion, *Back in No Time: The Brion Gysin Reader,* ed. Jason Weiss (Middletown: Wesleyan University Press, 2001) 79-94.

many duplicate lines. Gysin read it on BBC Radio in 1960. According to his own statement, it garnered the second-lowest approval rating of any broadcast. (Only W. H. Auden reading to Benjamin Britten's music did worse.) Gysin went on to permute the phrases "JUNK IS NO GOOD BABY," "KICK THAT HABIT MAN," "NO POETS DONT OWN WORDS," and "I DONT WORK YOU DIG." Gysin was one of a few artists and writers, working in different languages, who used a computer to permute phrases.[16]

Simple though Gysin's system is, there are exhaustive systems that are even simpler. Jörg Piringer's *Unicode* (2011) is a computer-generated digital video that runs for more than 30 minutes and shows every printable character in a large range of Unicode, a character set that is meant to be universal, at the rate of one character per frame. Enumerating all of the possibilities is like progressing through the alphabet, although in this case every current alphabet, and every current writing system, is included, and the video shows the unusual beauty and surprising extent of a standard method of encoding characters. *Unicode* relates to various poetic presentations of the alphabet by Aram Saroyan, Ian Hamilton Finlay, and others,[17] but because of its computerized means of production and its engagement with a coding scheme for the computer, it also engages with more complex enumerative systems.

Although the exhaustive system is really just presenting the full set of possible texts that the sampling system chooses from one a time, the effect of having everything presented at once is quite different. This is an effect that Gysin and Sommerville's radio listeners no doubt felt very keenly. Whether pleasing or not, the exhaustive system has a different poetics that showcases extent. The reader is not fooled into thinking that the possibilities are infinite, but may be impressed at how many texts result from the application of a single, simple rule.

An excellent example of a graphical exhaustive system, one which operates in time, rather than space, is John Simon's *Every Icon.* This Java applet from 1997 has never completed a run, but if and when it does it will enumerate every possible 32x32 monochrome icon. At current processor speeds, it will take several hundred trillion years to finish.[18]

Second Dimension: Static vs. Dynamic Source Texts

Digital conceptual writing has sped up and expanded since its early decades, working upon ever-larger data sets and exploring new crannies of our now-pervasive computers and networks. While early systems drew on relatively small distributions of data that were often included in the program itself, recent systems have been more open, and dynamic, in a variety of ways. They either invite user input directly or trawl the network for texts.

The examples discussed so far have used static stores of data. A program that uses data collected from the network, but keeps this data in a static store, is Poemitron, developed by Michelle Gay and her brother, Colin Gay. The version of this program that is part of *spampoet* uses a set of spam messages that have been collected. Although there is no live connection to the network, the language involved could not exist without the network and its commercialization. The set of data that is the basis for this work does not consist of human-authored texts (haiku, love letters) but texts that have been robot-generated for today's Internet.

Systems that use dynamic source texts include Bill Kennedy and Darren Wershler's *The Apostrophe*

16 Funkhouser, *Prehistoric Digital Poetry,* 40-42.

17 Dworkin, Craig and Kenneth Goldsmith, introduction to "Suicide" by Louis Aragon, *Against Expression: An Anthology of Conceptual Writing,* () 50-51.

18 Simon, John F., "Every Icon Project Page," 1997, <http://www.numeral.com/eicon.html>. Mirapaul, Matthew, "In John Simon's Art, Everything Is Possible," *The New York Times*, April 17, 1997, <http://www.numeral.com/articles/041797mirapaul/041797mirapaul.html>.

Engine and *The Status Update*, both of which plunder messages from the Web and from a social network and use them to produce texts directly for the "end reader." These systems have been used to write books[19] but have also done service on the Web,[20] where they have been invoked directly by readers.

Eric Elshtain and Jon Trowbridge's Gnoetry[21] is a different sort of system – a tool for authors seeking to statistically transform different source texts, usually prose, into verse. It is one example of a poetry generator that uses conventional statistical text techniques, more sophisticated than simple combination, and is intended for use by (cyborg) authors rather than readers. The typical use of this system has been with Project Gutenberg e-texts. For instance, the chapbook *The Dublin of Doctor Moreau* was created by Eric Elshtain using the system, James Joyce's *Dubliners*, H. G. Wells' *The Island of Doctor Moreau*, and certain formal constraints and indicated preferences.

Neither networked data nor the ability to accept text from the reader is necessary for conceptual computer writing, but many systems have used more open, dynamic capabilities to good effect. *The Status Update* jams two poets' Facebook RSS feeds up against Wikipedia's list of dead poets, conflating the transient and contemporary with the canonical and encyclopedic. It would be hard to create the same effect in a closed, stand-alone system with its own list. Nevertheless, there is plenty of evidence that non-networked, non-interactive systems can also work on language in interesting ways. The compelling diversity and repetition of *A House of Dust* is one example; the curated spam selection of the Poemitron-driven *spampoet* is another.

Third Dimension: Simple vs. Elaborate Systems

Systems for writing by computer can be straightforward, easily described, and simple or even trivial to implement. Or, they can be very elaborate, with architectures and subsystems that each warrant explanation.

Jim Carpener's Electronic Text Composition (or ETC Poetry Engine) is on the more elaborate side; Carpenter has presented about the system's architecture and development, backing his discussion with diagrams on PowerPoint slides. The system uses a relational database constructed from the British National Corpus and a great deal of code, only some of which has been released. (Gnoetry, in contrast, is a free software system, with source code that is freely available.) Electronic Text Composition has been used for various hoax purposes. The poems it generated were submitted to literary magazines, for instance, and the resulting rejection letters or acceptances were collected in binders.[22] Also, the system was used to generate *Issue 1*, edited by Stephen McLaughlin and Jim Carpenter, which consisted of 3785 pages of poems, all computer-generated using no special parameters and all attributed to living poets. A riot ensued wrecked the theatre.[23]

A number of the systems already discussed are quite different from Electronic Text Composition in terms of their complexity. The mechanism that underlies *about so many things* can be easily implemented in two lines of code in many programming languages. The principle underlying Gysin

---

19 Kennedy, Bill and Darren Wershler-Henry, *Apostrophe*, (ECW Press, 2006) and Kennedy, Bill and Darren Wershler, *Update*, (Snare: n.d.).

20 At <http://apostropheengine.ca> and <http://statusupdate.ca>.

21 Publications created using Gnoetry are available at <http://www.beardofbees.com/gnoetry.html>.

22 Carter, Erica T, "Erica T. Carter: The Collected Works," Slought Foundation, June 11, 2004, <http://slought.org/content/11227/>.

23 There is no comprehensive write-up of the *Issue 1* controversy, and no room to attempt one here, but see, for instance, Courtright, Nick, "Because I Can't Not Comment: The Issue 1 Controversy," *Tier 3*, October 13, 2008, <https://tier3.wordpress.com/2008/10/13/because-i-cant-not-comment-the-issue-1-controversy/> and the pages linked from there.

and Somerville's permutation poems is a simple mathematical one that can also be coded up in two lines, for instance, in Python. Although they use a more elaborate template, *Loveletters* and *A House of Dust* are also straightforward in terms of how they function, requiring no architecture diagram and no explanation of subsystems.

An important difference between simple, early projects and recent projects that embrace minimality and simplicity is the context of computing. Early projects showed us the delightful ways that "giant brains," early mainframes and then microcomputers, could work upon language; a simple program was enough to demonstrate this. Contemporary simple systems show us that the computer – networked to social sites and other sources of data, capable of generating amazing graphics and sound, able to work on massive stores of information – can still do a great deal of interesting conceptual work on language using only a small amount of code. In other words, very simple systems for conceptual writing once showed us how pleasing and stimulating simple computations could be; now, they work to help us not to discover but to recover that sense, which is increasingly lost as we consider the computer to be a networked media player that is important *only* because of the network and the mass of data that exists on it.[24]

Fourth Dimension: One Medium vs. Multimedia

Many of the examples here are essentially all-text, although the exhaustive *Every Icon* is a graphics-only project. One can choose to produce a "monomedium" or "multimedia" project quite apart from where one's system lies along the three dimensions already described. Talan Memmott's *Self Portrait(s) [as Other(s)],*[25] for instance, is a system that presents images and pseudo-biographies side by side, with both sampled from distributions. Reloading (or clicking "next artist," which does the same thing) presents another self-portrait and another franken-bio of an artist. So, the system is sampled rather than exhaustive. Although it runs in a browser, it works on its own data set rather than using dynamic data from the network. It is of moderate complexity, more elaborate than *about so many things* or the permutation poems but not as elaborate as Electronic Text Composition. And, of course, it is distinguished from the projects discussed earlier by presenting both combinatorially generated images and combinatorially generated texts.

Other Dimensions of Conceptual Computer-Generated Writing

These four dimensions are only a start, and capture only some of basic ways in which conceptual computing systems to generate texts relate to one another. An aspect that has not considered in detail is whether and how existing components of a computer system are being repurposed. Paul Chan's work to bend fonts into new sorts of writing machines, and to replace letters with phrases, is a prime example of this compelling line of work. If the code in which the system is implemented is meant to be read and understood in relation to the text machine's function, another dimension is active. This is one seen in Páll Thayer's series of Perl programs called Microcodes,[26] for instance, and in the practice of

24 My own work in developing very simple conceptual writing systems includes the *ppg256* series of 256-character Perl poetry generators; and several programs originally created as 1k Python programs (*Taroko Gorge*, *The Two*, and *Through the Park*); and *Concrete Perl*, a set of four 32-character concrete poems in Perl. More elaborate, but still meant to be understandable, is the potentially exhaustive system *Sea and Spar Between* that Stephanie Strickland and I created based on the writing of Melville and Dickinson. These systems are all linked from the main page of my site, <http://nickm.com>.

25 Memmott, Talan. "Self Portrait(s) [as Other(s)]," *Iowa Review Web,* April 2004, <http://iowareview.uiowa.edu/TIRW/TIRW_Archive/tirweb/feature/memmott/index.html>. Also available in the *Electronic Literature Collection*, volume 1, at <http://collection.eliterature.org/1/works/memmott__self_portraits_as_others.html>.

26 http://pallit.lhi.is/microcodes/

obfuscated coding.[27] So, this discussion has merely sampled, and not exhausted, the possibilities for computer conceptual writing.

Computer as Collaboration Machine

Computer writing with a conceptual basis has provided exhaustive permutations (Gysin and Sommerville), has extended to embrace the whole Web (Kennedy and Wershler's *The Apostrophe Engine* and *The Status Update)*, has used statistical methods and machine learning techniques (Elshtain and Trowbridge's Gnoetry), and has employed elaborate software architectures for hoax purposes (Jim Carpenter's system, used by editors to produce *Issue 1*). As varied as these writing practices are, they are also very often deeply collaborative – as these four examples show – and often invite collaboration in ways that other conceptual writing practices are not.

One common configuration for collaboration involves a known artist or writer working with a programmer: for instance, Ian Sommerville on the permutation poems, James Tenney on *A House of Dust*, Colin Gay on Poemitron. Even if the original team is careful to share credit, the programmer's name can fall by the wayside as the work is discussed in the context of other projects by the "first author." But the computer and the network do invite collaboration, not only in the "writer and programmer" mode but in other ways as well. A team of collaborators who are all technically and poetically engaged can use the networked computer to realize a project together even if they are not co-located, for instance.

Is There Non-Conceptual Computer Writing?

Creating computer text generators means mathematically modeling the production of language. It is an activity that seems inherently predisposed to conceptualism. Nevertheless, certain attitudes and goals can lead writer/programmers to develop systems that are very classically inclined or that even have romanticist rather than conceptual leanings. Haiku generators that are built in complete earnest, or with the goal of imitating human writing for social science or computer science purposes, can be instructive to look at (because of how many different styles of writing are produced within this form) but ultimately derive more from cliché rather than from a new and compelling idea that is formulated in terms of computation and the network. The output of such systems is, like conceptual writing, writing that does not need to be read – but in this case, it is for all the wrong reasons.

On the other hand, there are plenty of projects that employ compelling concepts but don't originate from an artistic or literary practice. On the social news site Reddit, a program called Haikubot detects when someone has posted a message that can be recast as a 5-7-5 syllable haiku; it responds by reframing and reposting the text to make a fresh and often hilarious connection between everyday writing and a classic poetic form. Molière's character Monsieur Jourdain discovers that he has been speaking prose all his life; the posters on Reddit discover, now and then, that they have been posting in verse. Just as Berne Porter found poems in everyday advertisements, forms, and other texts, people today are able to now find entire poetry generators that are feral – systems that were fashioned for other purposes but participate in the projects, inquiries, interventions, and hoaxes of conceptualism.

27 Montfort, Nick, "Obfuscated Code," *Software Studies: A Lexicon*, ed. Matthew Fuller, (Cambridge: The MIT Press, 2008) 193-199.