The Intrinsic Mutability of Code Poetry Uncovers New Notions of Poetic Design

CHRISTIAN IÑIGO D.L. ALVAREZ
christianinigo.alvarez@gmail.com

ABSTRACT

The paper focuses on code poetry and the need to extend current literary modes of analysis to accommodate aspects of digital poetry not previously found in written poetry. In the first section, a basic discussion on the difference between written poetry and code is made, which argues that while there are inherent differences between writing poetry and writing code, the syntax of modern programming languages is getting closer and closer to regular English sentences making the code a possible substrate for poetry. The transformative nature of programming languages makes them an interesting substrate. This leads to the question of how one could use current literary frameworks to analyze poetry that's created using a programming language, bringing with it elements not previously available in written text. What follows is a discussion of Turco's four levels of poetry, leading up to the extension of the framework with the addition of a fifth level--"the event level," which seeks to take into account the additional features of the digital substrate without breaking the existing modes of analysis of the written text.

KEYWORDS

digital poetry, code poetry, electronic literature, new media, programming languages

A Note to the Reader:

"A Conversation with the Lord" and "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" are texts fused into user interface elements created by different programmers. Being open source, modifying the HTML file, JavaScript file, and CSS file is allowed and encouraged (for it was through this exploration that these poems were born). For more information on copyrights and permissions for the usage, modification, and distribution of the source codes you may visit the following URLs:

For "A Conversation with the Lord"
http://codepen.io/shed_codepen/pen/obXoLL

For "r-p-o-p-h-e-s-s-a-g-r Caught in a Web"
http://codepen.io/anon/pen/vmVmbp

These two poems contain three parts: (1) the printed adaptation, which is what the poem would become if all the digital elements from animations to source code embellishments were removed to suit the traditional medium of poetry – paper; (2) the source code realization, which contains screenshots of the digital poems in action; and (3) the source code snippet(s), which show parts of the inner workings of the digital poems. The equivalent of (3) for poems in print could be thought of as the blueprint for the creation of the poems - the intent of the brain that led to the implementation of the written work.

```
//: Playground - noun: a place where people can play

import UIKit

let (myWords, evolve) = ("mutate", "come alive")
let thisPoem = ["fracture poems", "and code"]
for words in thisPoem
{
   print(words + " into a new plane of existence")
}
let me = "introduce a world beyond"
print(" where words " + evolve)
```

Introduction

Art in a person's mind never truly stays hidden. It manifests itself through the person's works, albeit in strange ways. As a programmer by profession, the poet in me found a way to manifest himself through code (for now, think of code as commands programmers give to computers), forging art of a different nature. I believe that poetry was able to find a way to materialize in code because there are underlying similarities to poetry and code--both in terms of creation and consumption.

When one creates a poem, one is said to "write poetry." When one creates a computer program/app, one is said to "write code." Both actions require not only a fundamental grasp on syntax and semantics, but also a grasp on certain conventions and techniques in order to create something that is worth reading (and using, for computer programs). The distinctions between code and poetry, however, become clearer upon examination of who the intended readers are, how these intended readers are meant to process poetry/code, and what happens after they have been processed.

For poetry, the intended readers, in general, are people who have the ability to understand and process the poem, not just in terms of the language used to create it, but also the literary techniques and tools employed. As the reader goes through a signifying process, the poem is instantiated in the reader's mind. When one studies poetry and its creation, then, one takes a closer look at the tools and techniques employed in order to facilitate this thought transfer from the author to the reader.

When one writes code, however, the primary readers are computers and other programmers, but the intended readers are the users. This means that code is created to be understood by the computers, first and foremost (through proper syntax), then programmers next (through consistent and standard programming conventions, other programmers read the code in order to modify it to add/remove a feature). Upon processing the written code, a literal transformation of the code happens in the computer, producing a concrete instantiation called a computer program/app. The user, then, becomes the intended 'reader' of this transformation, and another transformation happens in the reader's mind as he/she interacts with the program/app.

In reading poetry, what the poet has written is what the reader sees. For code, however, if one does not have access to the raw form of the text (the source code), the conversion of code from text to app means that the user 'reads' the processed form of what the author has written. The code has undergone changes that can make its raw form (text) look almost unrelated to its instantiated form (app). In short, unlike written poetry, code can go through literal transformations that result in a completely different text; hundreds of lines of code could very well transform into a one-line sentence that reads "Hello World!"

If one has access to the raw source code, however, what's interesting to note is that programming languages, particularly modern ones, read almost like normal English sentences with the aesthetics of mathematical formulas (i.e. the use of parenthesis, brackets, etc.). If a programming language, with its own sets of rules for syntax and semantics, allows for the creation of poetry in its raw form, as well as its realized form, how does one "read" and "write" code poetry?

Furthermore, if one is to employ literary techniques used in writing poetry in order to write code, how does this fit into the current literary frameworks used to analyze poems on the written page?

Also, since there is a literal transformation that happens to the text from code to app, can the current literary frameworks be sufficiently used to analyze both the raw and mutated texts in the substrates they are planted in, given that the digital space contains elements not previously present in the analog/written space? These are the issues that this paper aims to explore.

Before going any further, one must make the distinction among: (1) program's source code[1], (2) the output that's meant for the programmer, (3) and the output that's meant for the user.

The first is meant to be written by the programmer, and then understood primarily by the computer, then secondarily by fellow programmers. The second is essentially written by both the programmer and the computer - they often serve to inform whoever is programming what is going on while the program is running. The third is what the user sees - the realization of that source code; in everyday life, these are the apps in one's phone or the websites one visits. In this article, I shall focus on (1) and (3), taking into account that a lot of poets are not programmers.

The realization of this source code depends on the type of language used. For instance, hypertext markup language (HTML)[2] is a language meant to tag text for the computer to know how it should display certain content on a user's browser. Below is a poem, titled "Seashells," I made using html:

```
<!DOCTYPE html>
<html>
<head>
    <title>Seashells</title>
    <br>I am a <!--broken--> seashell,
    <br>white, clean, devoid of algae
    <!--of green, brown, or purple colors,
    colors I yearn to have on my surface
    for --><br>fish <!--to--> come <!--and graze at my exterior, not just -->to look at me
    <br>as I tumble on the dark brown shore <!--and leave, for I've nothing to offer-->.
    <br>
    <br>my mantle rests in my shell<!--no longer-->.
    <br>
    <br>I am a <!--broken--> seashell,
    <br>carried <!--and enslaved -->by the tumbling waves,
    <!--new cracks with every collision
    with rocks, driftwood, and sand -->
    <br>flying high and crashing low
    <br>to the rhythm of the <!--violent, shallow-->sea.
    <br>
    <br>Oh, how I wish <!--to be pulled into-->
    <br>the bottom of the sea,
    <br>with its troves and trenches <!--undiscovered and untouched-->,
    <br>creatures that roam free,
    <br><!--to-->see what awaits them - <!--that they may steer clear of-->
    <br>the bright and bubbly shore.

</head>
<body>

</body>
</html>
```

Figure 1: "Seashells" source code

Browsers read the tags as instructions on how to display certain elements in the code. For instance, the comment tag <!—<text>--> tells the browser to ignore the text it contains. These comments are for programmers to be able to give meaningful descriptions and comments on the code they make so that other programmers can collaborate and make necessary modifications easier. In the

1 In the field of programming, the text presented is called the *source code*. A source code is essentially a set of instructions for the computer to perform.

2 Technically, HTML is not a programming language, but the ability to mutate the text from one end and cause a ripple of changes on the other is what made me consider its inclusion as a part of the code poetry ecosystem.

poem presented above, however, it is used to change the way the poem is displayed in the browser. This is how the code above looks when opened by a web browser:

I am a seashell,
white, clean, devoid of algae
fish come to look at me
as I tumble on the dark brown shore.

my mantle rests in my shell.

I am a seashell,
carried by the tumbling waves,
flying high and crashing low
to the rhythm of the sea.

Oh, how I wish
the bottom of the sea,
with its troves and trenches,
creatures that roam free,
see what awaits them -
the bright and bubbly shore.

In this version of the poem, the texts inside the comment tags do not appear. The source code, then, has gone through a transformation that results in a different text.

In the first poem presented, instead of a browser interpreting the source code, an integrated development environment (IDE) called Xcode was used in order to transform the source code from sets of instructions for the computer to perform (source code), into the execution of those instructions (realization of source code). Below is a screenshot of the poem in Xcode:

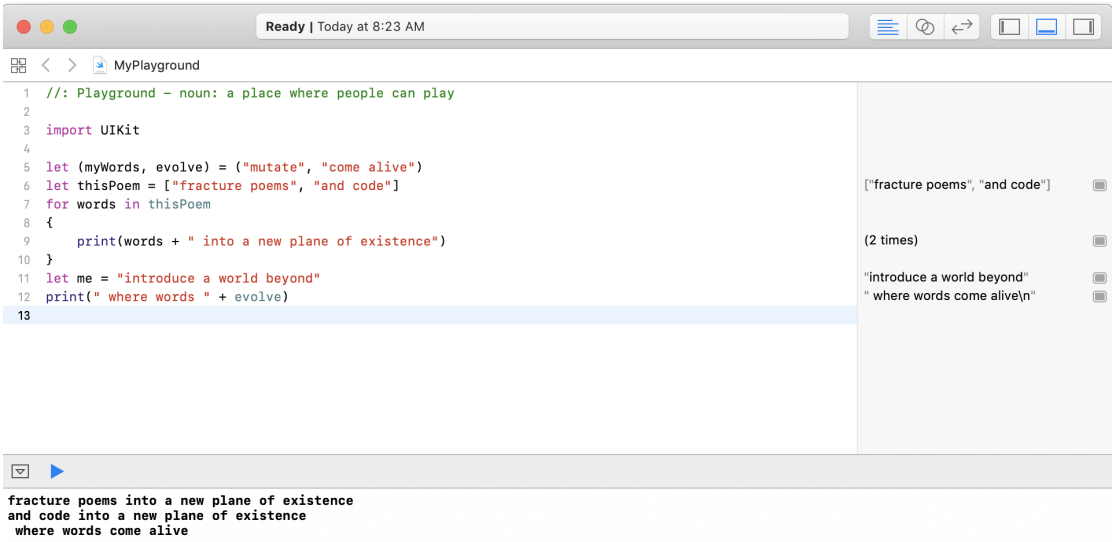

Figure 2: Xcode Screenshot 1

The visible result, as seen in the lower pane of Xcode, looks quite different from the source code:

fracture poems into a new plane of existence
and code into a new plane of existence
where words come alive

Like the Seashells poem, even though this poem was written once, it essentially mutates into another poem.

Xcode also provides a *Playground Mode* which shows a results panel on the right side of the source code, giving the programmer some insights into the inner workings of the source code. This creates yet another mutation of the original poem, as shown below, which demonstrates that the poem, though created only once, reveals three versions of itself shown by the IDE.

["fracture poems", "and code"]
(2 times)
"introduce a world beyond"
" where words come alive\n"

If one is to strip off the programming code syntax, one also effectively strips off the context of the medium that houses it. In fact, removing the programming syntax on the source code completely changes all the instances of the code, as shown below:

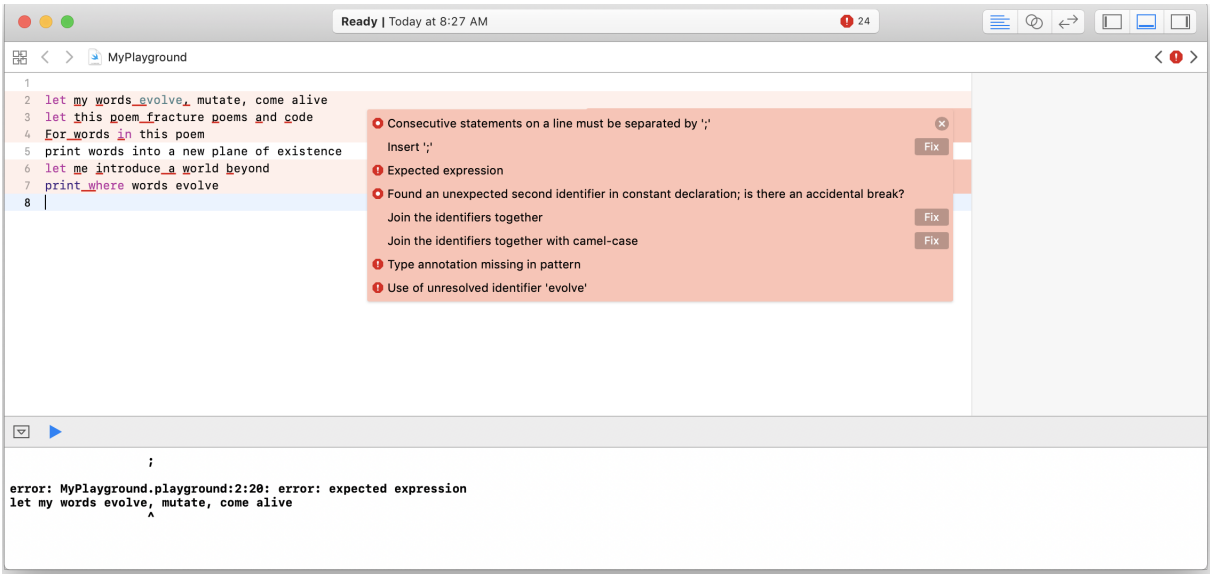

Figure 3: Xcode Screenshot 2

Notice how the pane on the right now appears blank, and the pane at the bottom contains a completely different text. Just as paintings viewed from one's phone offer different emotional and intellectual experiences compared to a more intimate encounter in a museum, retrofitting a poem written as source code (or a source code written as poetry) into plain text essentially affects the signifying process, as the output text(s) have changed due to the transformative nature of the medium (the IDE).

Poems created in the digital domain, which are quite possibly intimately connected and rooted in certain digital substrates (poems written using a specific programming language, for instance, will only be understood by specific software), contain elements that play a significant role in the signifying process. The first element shall be the most obvious: the visual element. For instance, the line: let (myWords, evolve) = ("mutate", "come alive") contains visual elements, the existence of which are necessitated by the adherence to the syntax of a specific programming language (Swift), and the appearance of which are determined by the program used to view the source code. The second element is the relationship between the raw text (source code) and the resulting text. The last element is the interaction between the reader and the output text and/or the source code. The source code, however, can be withheld from the reader since conversion of text from source code to output is a one-way process.

Because of this, code poems resist being read purely as plain text. Some written poems, in fact, particularly concrete poems, do seem to foreshadow the resistance of conversion of visual language into standard text through the intentional distortion of its typographical layer (a more in-depth discussion will be done on this later).

Similarly, the source code of the "Seashells" poem could be seen as a form of visual poetry, and its output, a written, printable lyric poem. If the source code is stripped off of its tags, the output also changes, resulting in an identical source code and output text. While one might argue that the two poems, while containing similar words, are semantically different and are therefore effectively two different poems, the relationship between the source code and the output, as well as the interaction of the reader with them, are actually integral to the central idea of the poem - the hidden texts that only a few people can see. By taking the poem's materiality as a part of the poem itself, it becomes possible to view the source code and the output as a single poem that is simply able to mutate based on the medium used, without disregarding the possibility of it being two separate poems. A programming text editor like Notepad++ shows the source code, and a browser like Google Chrome shows the realization of the source code.

If one, then, would like to take the materiality of the medium of the poem into account, as well as the mutations and distortions it can cause to the text, how is such a form of poetic writing understood, appreciated, or interpreted given the current modes of literary analysis?

Lewis P. Turco, an American poet and scholar of formal verse, defines four elements of poetry: *the typographical, the sonic, the sensory,* and *the ideational* (4). The typographical level is the first layer that readers see; it's the spatial aspect of the letters in the poem. Next is the sonic level, which comes after the letters are read and converted into units of sound. When these units of sound converge into words, images, thoughts, and emotions are evoked; this is the sensory level. The last level, the ideational, is the agglutination of these words, images, and thoughts into larger ideas and themes.

If a poem written originally as a source code can change into a different poem through the digital medium, therefore essentially producing a morph of itself, this poem can only be analyzed using Turco's framework if done separately on each text. The framework makes no allowances for text that is able to transform, nor could it take into account the very act of transforming the text from its raw form to its output, as well as the interaction of the user with the resulting text and the raw text, wherever applicable. If the visual, transformative, and interactive aspects of a code poem are central to a poem's conceptualization and creation, then an analysis with an assumption of the poem having a static typographical layer can significantly alter the signifying process, therefore affecting the traversal of the reader through the rest of the layers of poetry (sonic, sensory, ideational). With this, I am calling for a new level brought about by the nascent field of programmatic poetry, which I shall call the "event level." This, I believe, allows us to study and analyze poetry beyond the current limits of Turco's framework. To understand how this level helps provide a better interpretation of emerging digital poetry, I shall discuss the effects of the manipulation of the medium of texts on how the poems are experienced, perceived, and created. This manipulation comes either actively or passively from the materiality of the medium as it may either allow the reader to manipulate the text, or it could change the text without any user input.

Text: A Medium for Sound

Walter Ong, in exploring how the transition from orality to literacy affects culture and the human consciousness, reiterates the fundamental linguistic notion that written words are not direct signs, like spoken words or iconographic signs that represent ideas, but are mere attempts to represent the sound of spoken words (74).

Sound and text, however, belong in different domains--sound is in the temporal domain, and text is in the spatial domain. In attempting to represent sound through text, one effectively moves

sound from one medium to another. This may incur a kind of loss. For instance, digital files contain discrete rather than analog representations of data. Reality, however, is intrinsically analog, and so one has to capture analog data in digital form, therefore truncating some data in-between sampled points. Humans, however, do not notice this because the human senses automatically interpolate these data points. This means filling in the tiny gaps between these data points so that images in computers, for instance, which are made up of hundreds of thousands of pixels, are viewed as one whole image rather than a bunch of pixels with different colors.

For the purposes of this article, the act of transporting meaning from one container to another shall be referred to as "porting." In the context of programming, "porting" is the act of translating source code from one programming language to another. This term fits this article better than the term "translation," as the term "porting" covers the possibility of languages not existing or working properly with another medium. For instance, the modern programming language called *Swift* may only be used to create applications for Apple devices (Mac, iPhone, iPad, etc.), therefore a Windows machine will not be able to compile and understand Swift at all. In fact, some functions in Swift may not have any equivalent in another language, which, in effect, prevents a complete translation of a Swift source code into another programming language. While this is the case, however, this does not stop application developers from creating applications that claim to have almost exactly the same functionalities in Apple and Android devices--both of which use completely different languages (Swift and Java, respectively). One might call this a form of interpolation, as the Apple and Android app, though written in completely different languages, are often considered equivalent apps. As one might expect, some features and behaviors of these two apps will differ, albeit in minor ways, from the way tapping a button is animated, to the types of fonts used and the available features of the app.

The resistance of programming languages to be ported completely into another programming language, and the role of the programmer to minimize the difference such that the user effectively interpolates the gap (therefore effectively disregarding any tiny differences in terms of button size, animations, etc. and creating an illusion that the realizations of the source codes, though coming from different languages, are equivalent), is integral to the premise of the need to study the effect of the materiality of the medium on the language it contains.

Going back to the premise of written words as representations of spoken words, Ong also states that a sound is an event that "resists reduction to an 'object' or an 'icon' " (157). He elaborates on the resistance of sound to reduction through immobility:

> There is no way to stop sound and have sound. I can stop a moving picture camera and hold one frame fixed on the screen. If I stop the movement of sound, I have nothing—only silence, no sound at all. All sensation takes place in time, but no other sensory field totally resists a holding action, stabilization, in quite this way. Vision can register motion, but it can also register immobility. Indeed, it favors immobility, for to examine something closely by vision, we prefer to have it quiet. We often reduce motion to a series of still shots the better to see what motion is. There is no equivalent of a still shot for sound. An oscillogram is silent. It lies outside the sound world. (32).

Written texts, which belong to a spatial medium, and oral texts, which belong to a temporal medium, present a possibility of loss or change of information when converted from one medium to another depending on the capabilities of the origin and destination medium. For instance, choirmasters will have different interpretations of a single sheet of music even with the musical symbols (e.g., crescendo, pianissimo, etc.) and may ultimately end up performing a drastically different version of a piece played hundreds of years ago. This is because musical symbols serve only to approximate the sound produced and are therefore subject to misinterpretation. Written texts and symbols help readers

approximate sound, but because to date, there is no lossless way of converting sound to written text, there remains discrepancies between the originally conceived sound and the sound ported from text.

This is not to say, however, that the transition from orality to literacy has resulted in a far less rich medium. Just as iOS and Android have their own tradeoffs brought about largely by the differences in language and medium, porting text from the oral to the written domain also results in new capabilities (such as visual typography) in exchange for the old ones (such as precise tonal contours and meters). Ong further asserts that this movement of speech from the oral-aural to vision changes the very nature of speech and thought (83), as speech and thought, once only in the confines of the temporal domain, can now be recorded, viewed, and modified in the spatial.

An analysis of e.e. cummings' poem, "r-p-o-p-h-e-s-s-a-g-r" (CP 396), serves to illustrate this point:

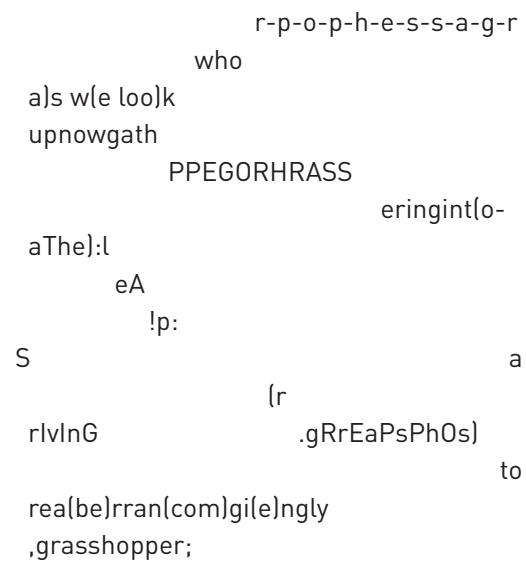

Figure 4: r-p-o-p-h-e-s-s-a-g-r by e.e. cummings

The poem is firmly grounded in the spatial domain and resists getting ported to the temporal domain of sound. One can test this assertion by attempting to read aloud the whole poem. The poem produces this effect by creating a unique placement of letters and punctuation marks, such that the reading experience becomes almost non-linear. This visual typographical clutter, however, is a necessary aspect of the poem, as it is meant to disrupt a seamless porting from the typographical layer to the sonic layer. cummings, in fact, wrote, "[N]ot all of my poems are to be read aloud—some . . . are to be seen & not heard" (*Selected Letters* 267). As one goes in a linear manner from the first element of Turco's framework (typographical) to the last element (ideational), the loss of the second element of Turco's framework forces the reader to create an artificial sonic layer as a bridge between the typographical and the sensory layer. This means forcibly porting the written text into oral text. For instance, "rea(be) rran(com)gi(e)ngly," with the artificial sonic layer, can now be read as "become rearrangingly or rearrangingly become." In an effort to produce a clean artificial sonic layer, the reader has to jump back-and-forth between the letters, mentally removing extraneous punctuation marks, rearranging the letters, etc. The very act of mentally arranging these letters and removing punctuation marks, then, provides an artificial sensory layer that links the jumping of the reader's eyes from one letter/ punctuation mark to another, giving off a sense of the chaotic motion of a jumping grasshopper.

In effect, it is through the engagement of the reader with the text that one was able to link the typographical clutter to poetic thought. The reader, in an effort to do this, has created an artificial sonic layer from the written text in an attempt to retrofit the mess of letters and symbols into what is

syntactically and semantically acceptable.

Written text is able to represent utterances by attempting to capture sound through morphemes. This demonstrates that the manipulation of the written text through visual iconicity and use of syntactic distortions affects the sound that the text has captured via morphemes, effectively creating new ways of producing sound that could not have been created in a primarily oral culture.

Mutating the Written Text Through its Medium

If e.e. cummings' "r-p-o-p-h-e-s-s-a-g-r" disrupted the porting of written text to oral text by distorting the typographical layer, it follows that distortion to the medium of the typographical layer creates a cascade of distortions on a sonic, sensory, and ideational level.

For instance, here are two images of my first draft of two haikus from a class that can also be read as one poem:

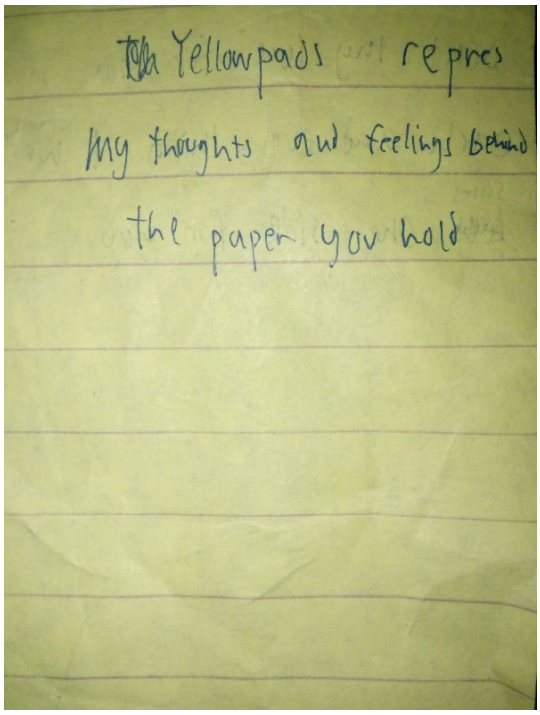

Figure 5.1: Page 1 of Yellow Pad Poem

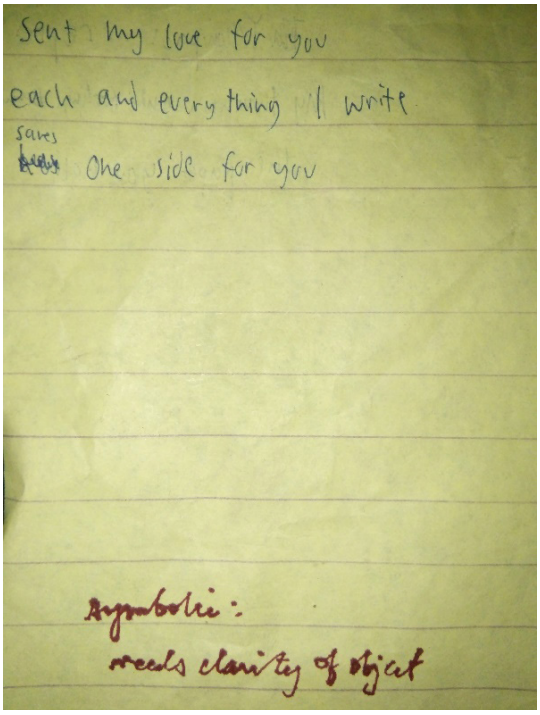

Figure 5.2: Page 2 of Yellow Pad Poem

Both texts are in a haiku form, but they are connected to each other such that the poetic lines can run from page 1 to page 2. This is why in the first page, "repres" is misspelled; it creates a sense of incompleteness that may entice the reader to look at the back of the page. On the second page, however, there's a teacher's note saying "symbolism: needs clarity of object," which, upon clarification, meant that he did not see the first page and assumed that there was only one poem. In that sense, by not being read, the idea of the haiku in the first poem was still realized, but it is in the very act of flipping the page and reading the two pages that the complete poem emerges. Porting this poem into typewritten text, however, removes the reference of the yellow pad, and if read in a computer, changes the nature of the engagement of the reader with the text (the reader no longer flips pages but scrolls through text). This means that changes and distortions to the medium that contains the text can invariably change how the reader engages with the text, and if the reader's engagement is integral to the poetic experience, it may very well enhance or ruin the experience. Interestingly, however, the only element of Turco's framework that could be used to analyze the changes and distortions to the medium is the typographical layer. However, because the typographical layer is spatial in nature and does not

take into account the engagement of the reader, analysis through this layer will be limited to separately analyzing the first haiku, the second haiku, then the two combined as one poem.

dn (teeg .The Emergence of the "Event Level" in the Study of Poetry

If distortions in the medium can cascade from the typographical level all the way down to the ideational level, then continuous distortions (not necessarily associated with loss, but with change) may result in a continuous string of distorted layers of poetry that resist being arrested in the spatial domain. This distortion, then, merits further analysis as it adds to what the medium can portray. A possibility of this exists even outside the digital domain. In the Yellow Pad poem, for example, it is through the flipping of the yellow pad that the poem emerges, and this action of flipping belongs to the temporal domain. Following the order of reading from left to right, the reader has to flip the yellow pad six times to form the large poem from the two haikus. The digital domain, however, possesses more possible distortions, through different kinds of animations and shapeshifting capabilities (i.e., in the compilation of source code). In the digital domain, where the user can interact with the text possibly resulting in continuous changes to the text (animations) or production of sound and other external stimuli, the poetic experience can no longer be confined to static text. The analysis of digital poetry, therefore, moves from the static page into the engagement of the reader and the materiality of the medium. The semiotic experience (the creation of new meanings) that emerges -- the continuous string of distortions and mutations that result from the engagement of the user and materiality of the medium -- lies in a new layer: "the event level."

Absorption and Anti-absorption

In the digital domain, the ability of code poem to shape-shift from one form into another inevitably foregrounds its form. This is what Bernstein refers to as antiabsorptive writing in his essay "Artifice of Absorption." An antiabsorptive writing is a kind of writing that foregrounds the materiality or artifice of the writing. Absorptive writing, on the other hand, is one that minimizes the materiality or artifice of the writing and foregrounds the content. According to Bernstein, the two aren't mutually exclusive; the extent of foregrounding the artificiality of a poem varies from poem to poem. He does, however, emphasize that the artificiality is a necessary part of a "poetic" reading, and that "[c]ontent never equals meaning" (10).

Most visual poems can be taken as an anti-absorptive form of writing, as the icons loaded with meanings are scattered across the page, while most lyric poems can be taken as an absorptive form of writing, as the typographical level is crafted to create a seamless transition into the sonic level, where the rhyme and meters are stored. Digital poems, on the other hand, because of the temporal aspects they can contain and manipulate, can also effectively manipulate its absorptive and anti-absorptive properties. For instance, a programmer poet could create a sonnet that changes the position of its words to form a picture with a press of a button and revert back to the original form with another press of a button.

The ability to traverse through different absorptive and non-absorptive states of a text results in an additional viewpoint that was not previously available to the reader in print -- a real-time transformation of the text. For visual poetry, this becomes a crucial aspect, as the text can now be manipulated not just in the spatial domain, but also in the temporal domain, therefore appearing, disappearing, changing colors, rotating, and performing other changes to its typographical layer as needed. For example, here is a digital port of e.e. cummings' "r-p-o-p-h-e-s-s-a-g-r" that I call "r-p-o-p-h-e-s-s-a-g-r Caught in a Web"

Figure 6.1: "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" Frame 1

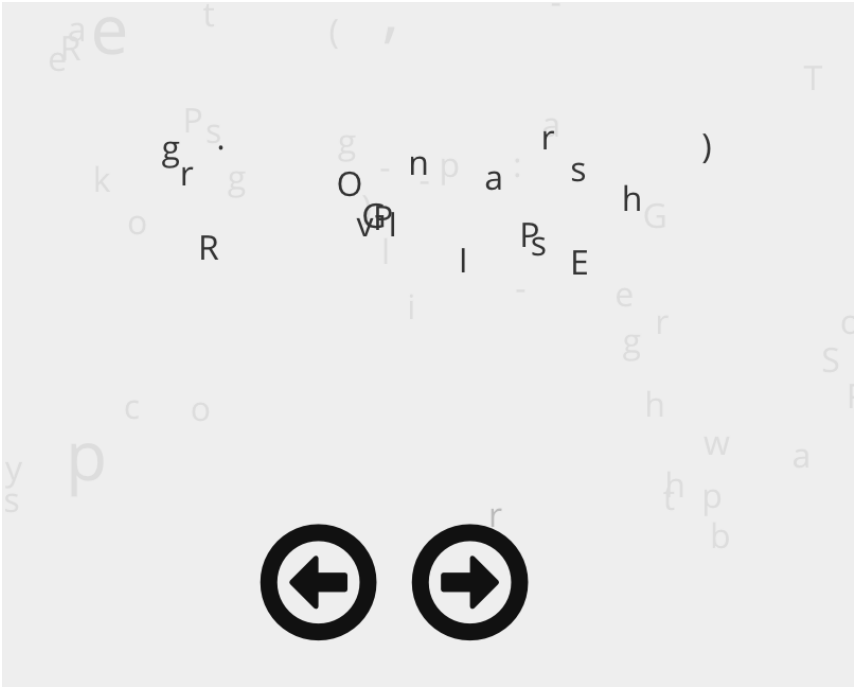

Figure 6.2: "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" Frame 2

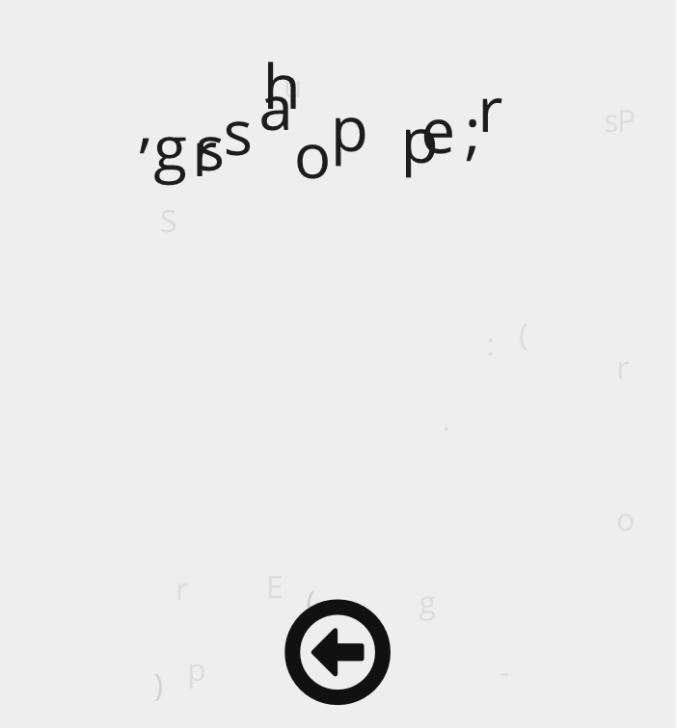

Figure 6.3: "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" Frame 3

And here are the code snippets:

```
$(document).ready(function() {
 /*
  * Main variables
  */
 var content =
 [
  {
   title: "",
   desc: "               r-p-o-p-h-e-s-s-a-g-r"
  },
  {
   title: "",
   desc: "            who"
  },
  {
   title: "",
   desc: "  a)s w(e loo)k"
  },
  {
   title: "",
   desc: "  upnowgath "
  },
  {
   title: "",
   desc: "           PPEGORHRASS"
  },
  {
   title: "",
   desc: "                  eringint(o-"
  },
  {
```

Figure 6.4: "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" Source Code Snippet 1

```
  title: "",
  desc: "  aThe):l"
 },
 {
  title: "",
  desc: "       eA"
 },
 {
  title: "",
  desc: "        !p:"
 },
 {
  title: "",
  desc: "S                          a"
 },
 {
  title: "",
  desc: "              (r"
 },
 {
  title: "",
  desc: " rIvInG          .gRrEaPsPhOs)"
 },
 {
  title: "",
  desc: "                        to"
 },
 {
  title: "",
  desc: " rea(be)rran(com)gi(e)ngly"
 },
 {
  title: ",grasshopper;",
  desc: ""
 }
];
```

Figure 6.5: "r-p-o-p-h-e-s-s-a-g-r Caught in a Web" Source Code Snippet 2

Instead of just one image formed by the scattered letters, the poem now plays an animation of scattered letters for every line, almost as if it was imitating the scattered movements of the grasshoppers, eventually resolving into the word "grasshopper" in the end. Take note, however, that porting written text into the digital domain involves a form of interpretation on the one who ports the text.

The progression of images shown by the poem could be seen as something similar to a video, except that for code, there is a possibility for the reader to choose the traversal of the text, and in some cases, change the possible traversal of the sequences by modifying the source code of the poem. There is also a possibility to introduce randomness into the poem, therefore producing an almost infinite number of paths for the user to traverse. For instance, the found poetry project entitled "Hi Ma'am Sir" by Adam David contains combinations of lines from Fast Food Fiction Delivery Volume 2, a sample of which is shown below:

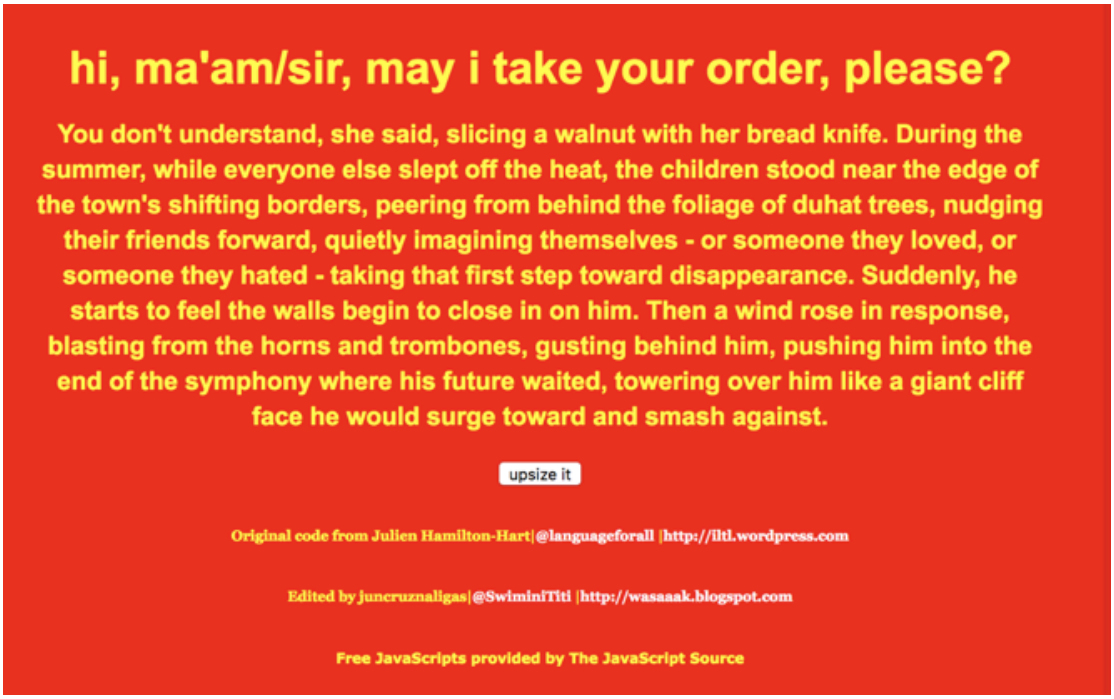

Figure 7: "Hi Ma'am Sir" Screenshot

While technically, there is a finite number of combinations for each line, it seems almost like an infinite number of possible texts could be generated. One could say, however, that the central idea of the project is not found in a single generated text. Rather, it's found in the very act of generating text – the interaction of the reader with the text becomes important in generating not just the text, but the meaning of the text.

For lyric/prose poems originally from print, however, new aspects and new ways to view and understand the poem could be added by porting it into the digital domain. As a lyric poet, I find that lyric poetry and code are compatible. Lyric poetry contains musical language that may suggest a form of movement, while code can actually show movements through sound, animations, or the transformation of the text, whether it's from the user, the medium, or both.

An example of a lyric poem that relies heavily in the ability of the medium to distort the typographical layer is "A Conversation with the Lord" (see Figures 8.1 to 8.3). This poem was built on top of code created by other programmers. The nature of these open-source source codes is that upon their creation, they are free to use, modify, and redistribute so long as the copyrights for modification, distribution, etc. are packaged with the code. With this, people could build upon each other's work, making changes as they deem necessary for what they plan to use the code for. This means that the origin of the source code might not be easily traceable unless it's placed in version control (version control is a software that allows programmers to save "snapshots" at certain points of the source code). This allows for collaboration and improvement of the source code by different programmers. This is not usual practice in digital poetry (as shown by the fact that Adam David's blog, shown above in Figure 7, has been taken down due to copyright issues in his found poetry project[3]). In fact, these source codes were originally meant to help people improve the look and feel of their websites. I do find, however, that it is a more accessible way to help non-programmers to create their own code poetry.

3 "Hi Ma'am Sir" is a hypertext found poetry project that consisted of a simple Javascript code that generated randomized combinations of sentences obtained from different short stories. The project was taken down due to copyright issues with sentences it used that it had taken from several short stories. What remains of the project can be found here: http://himaamsir.blogspot.com/. Since Google saves the recent states of the websites it finds, I was able to access and download the original project.

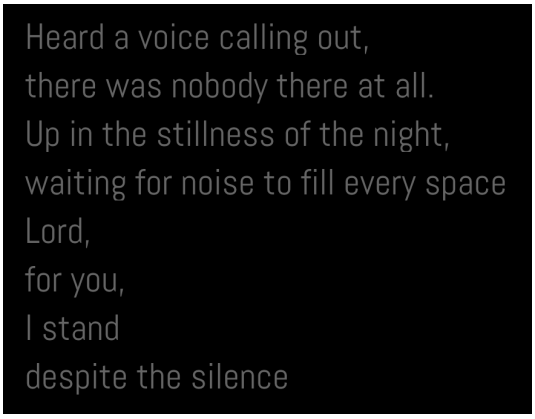

Figure 8.1: "A Conversation with the Lord" Frame 1

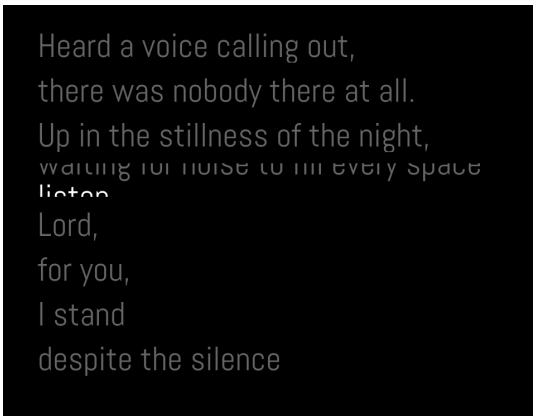

Figure 8.2: "A Conversation with the Lord" Frame 2

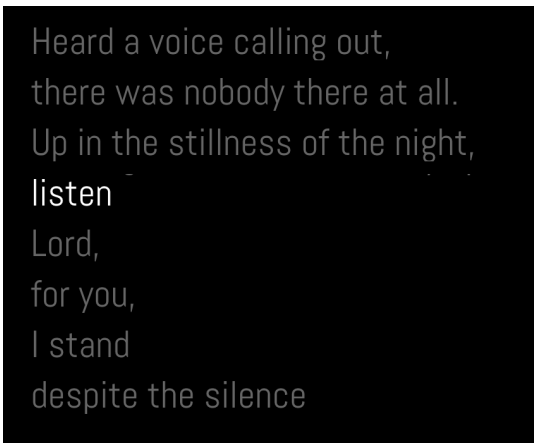

Figure 8.3: "A Conversation with the Lord" Frame 3

The poem relies on the interaction of the user through the cursor. As the curser hovers through the poem, the poem reveals lines that appear to be from the other side of the page through a flipping animation. The following are screenshots of the source code:

```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>A Conversation with the Lord</title>

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/meyer-reset/2.0/
  reset.min.css">

  <link rel='stylesheet prefetch' href='http://maxcdn.bootstrapcdn.com/
  bootstrap/3.3.5/css/bootstrap.min.css'>

      <link rel="stylesheet" href="css/style.css">


</head>

<body>
  <div id="app">
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">Heard a voice calling out,</span>
        <span class="m-flip_item">Listen,</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">there was nobody there at all.</span>
        <span class="m-flip_item">silence is pervasive</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">Up in the stillness of the night, </span>
        <span class="m-flip_item">stop</span>
    </p>
```

Figure 8.4: "A Conversation with the Lord" Source Code Snippet 1

```
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">waiting for noise to fill every space</span>
        <span class="m-flip_item">listen</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">Lord,</span>
        <span class="m-flip_item">for I am</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">for you,</span>
        <span class="m-flip_item">with you</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">I stand</span>
        <span class="m-flip_item">always</span>
    </p>
    <p class="m-flip js-flip" style="font-size: 30px; font-family:Abel;">
        <span class="m-flip_item">despite the silence</span>
        <span class="m-flip_item">until the end of time </span>
    </p>

</div>
  <script src='http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js'></script>

    <script src="js/index.js"></script>

</body>
</html>
```

Figure 8.5: "A Conversation with the Lord" Source Code Snippet 2

This poem was made with the engagement of the reader in mind; the flipping animation that is triggered when the cursor hovers over the lines of the poem represent a message from the other side ("the other side" having three possible meanings: one, being a simulation of the back of a written page; two, being the source code; and three, being God). The whole poem still makes sense even if hovering the cursor over a line replaces it with a line from the other sidee. If we take these line changes as an instantiation of another poem, one could see this poem as having ten instances -- one from the untouched state, eight coming from each line, and one from the whole source code. One could even combine the alternate lines and form yet another poem, though this would be considered an artificial typographical layer, as there is no immediate instance of this in the source code or output. The untouched state is the one the reader is guaranteed to see upon opening the webpage, but the rest is dependent upon the reader's choice of hovering the cursor over the lines of the poem, as well as viewing the webpage's source code to read the poem's lines combined, which when converted to plain text becomes:

A Conversation with the Lord
Heard a voice calling out,
Listen,

there was nobody there at all.
silence is pervasive

Up in the stillness of the night,
stop

waiting for noise to fill every space
listen

Lord,
for I am

for you,
with you

I stand
always

despite the silence
until the end of time

The poems I have presented so far (//: Playground, Seashells, r-p-o-p-h-e-s-s-a-g-r, Yellow Pad, r-p-o-p-h-e-s-s-a-g-r Caught in a Web, Hi Ma'am Sir, and A Conversation with the Lord) have aspects of antiabsorptive writing because of the degree to which they foreground the artificiality of the reading experience through the unconventional engagement of the reader with the text. The digital medium, however, offers a unique confluence of absorptive and antiabsorptive qualities by turning poetry from a non-changing text into a shapeshifting text that can shift back and forth between absorptive and antiabsorptive writing because of its ability to hold temporal data (whereas written text only stores visual data pertaining to temporal information that has to be decoded by the reader).

Ong, in fact, acknowledges the movement from the written word to cyberspace as a form of a "secondary orality," which is:

> …both remarkably like and remarkably unlike primary orality. Like primary orality, secondary orality has generated a strong group sense, for listening to spoken words

> forms hearers into a group, a true audience, just as reading written or printed texts turns individuals in on themselves. But secondary orality generates a sense for groups immeasurably larger than those of primary oral culture—McLuhan's "global village." (133)

The secondary orality brought about by the emergence of digital media allows events to be reproduced in precisely, for different audiences in different places, with new ways to engage with the text, as well as new ways for the text to manifest itself in. Poets, through the digital medium, gain access to the temporal domain to manipulate the written text in ways that have never been possible with static media, such as paper.

The digital manipulation of the non-absorptive qualities of a text gives the author more control over the poetic uptake of the reader. For instance, in "A Conversation with the Lord," I made constraints to what the poem does such that it only changes the line that the user's cursor is hovering over. In static written text, there is no way to selectively hide and reveal parts of the poem without having to tell the user, perhaps, to manually cover some of the lines and reveal only parts of it as he/she pleases. This creates a less seamless experience of the poem compared to the intuitive flipping of a line the user points at. The readers, through the response of the medium (in this case, the flipping of a line to reveal a different word/phrase that affects the whole poem), are also given access to what the poet intends to express. For instance, as a reader, one could see the flipping of the lines in "A Conversation with the Lord" as a way for the author to show how every line from the other side of a poem is an answer from the Lord, which effectively changes the prayer as the reader moves the cursor down through the lines. This idea arose from the notion of *illocutionary acts* developed by J.L. Austin and later refined by J.R. Searle. Austin notes that utterances can be characterized as: *locution*, *illocution*, and *perlocution*. Locution is described as "the words used to form the utterance and the grammatical *form* of the utterance expressing a proposition" (139). Illocution is described as "what the speaker (Sp) intends to perform" (139). Perlocution is what is achieved and is described as having twofold effects in the case of a pragmatically understood request (e.g., saying "it's cold" as a request for a jacket), "depending on (a) whether [the hearer] understands the utterance of the speaker and (b) if so, whether or not [the hearer] is actually willing to comply with the request" (139).

It was also noted that: "[t]he perlocutionary component of the utterance also highlights the importance of mental constructs in pragmatics: both [the speaker] and [the hearer] have certain beliefs that affect their *intentions* or *goals* in an exchange, as well as the effect of utterances" (139).

These ideas are applied to the pragmatics of speech, but on the basis that written text is ported sound (or speech, in this context), and a premise of poetry being a dialogue between the poet and the reader, these acts can be adapted to the study of poetry as:

> 1. Locution – The mediation (text) of the poet of the conversion from letters to utterances through prosody and syntax manipulation
> 2. Illocution – The images or thoughts that the poet (author) wants to convey through those utterances
> 3. Perlocution – The uptake by the receiver (reader) based on the cumulative effects of the utterances

Both the absorptive and nonabsorptive qualities of a text can be found in the locutionary effects of the poem. For the readers, though, the foregrounding of the materiality of a poem (the characteristics and capabilities of the medium that contains it) makes the illocutionary aspect more apparent (e.g., using animations in a specific manner, e.g., fast animations to illustrate turbulence or violence), and in limiting perlocution, allow a more guided reading of the text. The act of foregrounding the artifice of the reading experience can be likened to adding footnotes to help guide the reading of the poem the intended effect being a better conversion from written text to imagery and thought.

User experience (UX), which refers to a user's attitude towards a product, is a key aspect in programming. In this article, it shall be paralleled with perlocution, as it is concerned with the uptake of the user based on the cumulative effects of the elements displayed in a program or a webpage. Programmers in the field of UX study how users respond to the user interface of programs, web pages, and even operating systems. This is why programs are becoming more and more "human" in terms of how they respond and interact with the user; UX programmers' line of focus is creating interfaces that users are used to (such as having "File" menu on the upper-left portion of applications that handle files). A poem that's natively digital (created using digital technology, and consumed using digital technology), then, must adhere to these standards. The interactive nature and well-studied perlocutionary effect of the digital medium reduces the tendency of the reader to become hostile or bored with the texts they encounter, therefore significantly reducing the antiabsorptive effects of foregrounded materiality.

The Emergence of the "Event Level"

In fitting digital poetry into Turco's layers of poetry, I propose a new layer -- a layer that describes the engagement of the user with the text, and the medium that allows this engagement to happen. I propose that these be collectively known as the "event level." With the emergence of poems that are able to respond and change with user interaction, there needs to be an expansion of literary analysis in order to accommodate the materiality of texts, the possible and/or intended interaction between the reader and the text, and the actual interaction that takes place. I believe that while this may have been already present in concrete and pattern poetry (with the form of interaction being the non-trivial way in which the reader engages with the text – by reading a scrambled letter combination, flipping back and forth a page, etc.), the transition to a secondary orality warrants this expansion because of the myriad of emergent potential methods of interacting with text brought about by the digital medium.

While the typographical layer of poetry contains the written text in the spatial context, the event level contains the mutations to the written text in the temporal context. This includes visual elements applied to the written text (whether it animates the text itself or produces an entirely different element through the text), sounds (produced by the text or with the text), haptic feedback (like vibrations from a mobile phone), and possible user interactions through different modes of input (touchscreen, voice, etc.). In essence, the event level is able to distort the typographical, sonic, and sensory layers directly through the temporal domain, and could thus be placed on the first layer:

1. Event
2. Typographical
3. Sonic
4. Sensory
5. Ideational

Or, as it is, by nature, temporal, it might very well not fit in properly in a standard list:

1. Typographical 1 □ <Event A> □ Typographical 2
2. Sonic 1 □ <Event A> □ Sonic 2
3. Sensory 1 □ <Event B> □ Sensory 2
4. Ideational 1 □ <Event A + Event B> □ Ideational 2

This second list considers the temporal nature of the event level and places it in-between changes from the layers. The first three layers can be directly modified by the event layer, but the fourth layer is only indirectly modified through the first three layers (hence the addition of events from the first three layers).

The layers could very well stack up to more than two, since the event layer can distort them more than once. With this new frame of analysis, I believe that digital poetry, or even other transmedial

poems that rely on the materiality of the medium and the interaction of the reader, can be better understood and analyzed, especially due to the transformative nature of programming languages.

Artists like Eduardo Kac have taken the liberty of exploring this aspect of materiality and interaction. His work called "Genesis," is described in his site as:

> …a transgenic artwork that explores the intricate relationship between biology, belief systems, information technology, dialogical interaction, ethics, and the Internet. The key element of the work is an "artist's gene", a synthetic gene that was created by Kac by translating a sentence from the biblical book of Genesis into Morse Code, and converting the Morse Code into DNA base pairs according to a conversion principle specially developed by the artist for this work. The sentence reads: "Let man have dominion over the fish of the sea, and over the fowl of the air, and over every living thing that moves upon the earth." It was chosen for what it implies about the dubious notion--divinely sanctioned--of humanity's supremacy over nature. Morse code was chosen because, as the first example of the use of radiotelegraphy, it represents the dawn of the information age--the genesis of global communication. The Genesis gene was incorporated into bacteria, which were shown in the gallery. Participants on the Web could turn on an ultraviolet light in the gallery, causing real, biological mutations in the bacteria. This changed the biblical sentence in the bacteria. After the show, the DNA of the bacteria was translated back into Morse code, and then back into English. The mutation that took place in the DNA had changed the original sentence from the Bible. The mutated sentence was posted on the Genesis web site. In the context of the work, the ability to change the sentence is a symbolic gesture: it means that we do not accept its meaning in the form we inherited it, and that new meanings emerge as we seek to change it. (Genesis)

Kac's work is categorised as a non-absorptive work, as the key element of this poem is the interaction or non-interaction of the participants to the DNA base pairs through an ultraviolet light. In this context, poetry becomes a form of performance art, where the art takes place in a temporal domain.

The digital domain is, however, unique in a sense that it can lie dormant in a source code or executable program file, and come to life upon the decision of the user. Because of this, digital poetry is able to reach more people compared to transmedial poetry that's reliant on specialized equipment.

Programming Languages as Fertile Poetic Ground

Stanford has created a code poetry slam program, which is a poetry competition that involves creating computer code that can be read as poetry. Stanford (Code Poetry Slam) defines Code Poetry as different things to different people, but quite possibly one of the following:

> a. Poems written in a programming language that are meant to be simply words, albeit heavily syntactically embellished
> b. Elegantly-written code within the constraints of traditional poetry, such as haikus or sonnets
> c. Code that generates poetry

In addition to the three possible definitions of code poetry, there are also two additional conditions that must be simultaneously met:

> a. Computers have to be able to understand the language (no syntax errors)
> b. The source code must be beautiful to read

Extending Stanford's definition of code poetry, code poetry could be dynamic poems that can change form according to certain factors, such as time of day, type of device used, etc.

The digital domain is a ripe literary substrate for poetry, allowing programmers to use the language of computers to create poetry that responds, poetry that grows, and poetry that almost seems self-aware.

Kac mentions his desire to escape the printed page and develop poetry that is no longer tangible but immaterial in line with the information age. He talks about how he wanted to develop "poetry native to the new cultural environment of digital global networks, with its dynamic data flux and distributed communication systems" (45).

With the intention "to push further the visual syntax and the rarefied lexicon that has been a main vector in experimental poetry since Cummings, Belloli, Dias-Pino, and many others" (Kac 46), Kac explored the digital world, which allowed the transference of the written text into quite literally, new worlds in the digital domain.

In a book titled, "10 PRINT CHR$ (205.5 + RND (1)); : GOTO 10," ten authors collaborate in dissecting a single line of code that also happens to be the book's title. The code is written in BASIC, which is an old programming language used by Commodore 64s. The line of code, when run, actually produces a fascinating graphical maze that ends only when the user inputs "CTRL+C," which forcefully terminates the program.

The book talks about how one could actually look at programming languages not just as formulas one feeds to the computer, but also an account of how the society using the language functions and interacts:

> Like a diary from the forgotten past, computer code is embedded with stories of a program's making, its purpose, its assumptions, and more. Every symbol within a program can help to illuminate these stories and open historical and critical lines of inquiry…[I]n the emerging methodologies of critical code studies, software studies, and platform studies, computer code is approached as a cultural text reflecting the history and social context of its creation. (3)

With this, I claim that poetry is an elevated use of human language that allows for an efficient encapsulation of ideas and emotions, which would then be evoked in the readers who are able to understand the syntax, semantics, and conventions of the poem in question. By using programming languages as a substrate for poetry, one effectively grounds poetry in a digital world that, I argue, is largely grounded on metaphors that link even the most complex human ideas to sparse lines of code.

Since the creation, development, and usage of a language all depend on the culture it is rooted in, the parallel drawn between programming languages and human languages also roots programming languages into the culture of the programmers who write the code, as well as the people who use the programs created from this code. Code, then, like human language, adapts or dies based on the needs of the people who write it (programmers) and the people who "read" it (users). It becomes invariably interlinked with culture, therefore making it an important piece to consider in attempting to understand the collective human condition, as Monfort et al explain below:

> …code is a cultural resource, not trivial and only instrumental, but bound up in social change, aesthetic projects, and the relationship of people to computers. Instead of being dismissed as cryptic and irrelevant to human concerns such as art and user experience, code should be valued as text with machine and human meanings, something produced and operating within culture. (8)

The digital substrates at which poetry could thrive in has become quite vast, from the simple Kindle book that can store hundreds of poems, to web pages and apps that offer a new level of interaction that merits an expansion of literary study.

Sent[i]ence

Digital poetry, with its dynamic and transformative medium, does not simply offer superficial visual flair. Rather, it introduces new viewpoints through the transformation of the source code into the output text, through transformations of the output through user interaction, or a combination of both. These new viewpoints bring about fresh and unique textual encounters that can branch out into new notions of poetic design.

De Leon, Jr. talks about the tendency of Filipinos to fill up space: "The common Filipino is a maximalist, filling up every available space with forms and things. It springs from an expressive exuberance deeply rooted in emotional sensitivity and the strong urge to connect." I believe that this applies especially to poetry – a weed that can grow from the tiniest cracks to the largest fractures. As a Filipino poet, I saw a crack in the sheet of paper – a possibility for the text to engage the reader in a different way. As a Filipino programmer, I saw a crack in the source code and the program output – a substrate for new forms of poetry to grow in. As a programmer-poet, I saw a fracture between the digital and the written domain, paving way for the expansion of poetic creation and study using a digital substrate, and the expansion of the machine's multimedia capabilities and understanding of human language with the study of poetry – the highest form of literature. In attempting to find Filipino code-poets, I found out that the hypertext projects and other code poems are mostly dead links; code poetry in the Philippines, to date, is virtually nonexistent (or at the very least, unsearchable). In creating these digital poems, I hope to become a cast for this fracture – an invitation for Filipino poets and programmers to fill in the gaps with their creations, and in doing so, push programmatic and poetic design to greater heights, to the rise of sentient poetry: poetry that almost feels, perceives, and responds – poetry that lives.

WORKS CITED

Bernstein, Charles. *A Poetics*. Harvard University Press, 1992.

Cummings, E. E. *Complete Poems 1913-1962*. Ed George J. Firmage. HBJ, 1972.

—. *Selected Letters of E. E. Cummings*. Ed. F.W. Dupee and George Stade. HBJ, 1969.

David, Adam. "hi, ma'am/sir, may i take your order, please?" himaamsir, n.d, http://himaamsir.blogspot.com/.

De Leon, FM, Jr. "Life as Art – The Creative, Healing Power in Philippine Culture." *In Focus*, 25 Feb. 2015, http://ncca.gov.ph/about-culture-and-arts/in-focus/life-as-art-the-creative-healing-power-in-philippine-culture. Accessed 21 May 2017

Kac, Eduardo. "Genesis." *KAC*, n.d, http://www.ekac.org/geninfo.html. Accessed 30 Sept. 2016.

—. *Media Poetry: An International Anthology*. Intellect Books, 2007.

Kagen, Melissa and Werner, Kurt. *Code Poetry Slam*, n.d, http://stanford.edu/~mkagen/codepoetryslam/. Accessed 20 Oct. 2016.

Mitkov, Ruslan. *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2002.

Monfort, Nick, et al. *10 PRINT CHR$(205.5+RND(1)); : GOTO 10*. The MIT Press, 2013.

Ong, Walter J. *Orality and Literacy: The Technologizing of the Word*. Methuen, 1982.

Turco, Lewis. *The Book of Forms: A Handbook of Poetics*. University Press of New England, 2000.

CHRISTIAN ALVAREZ is an iOS Developer at Travel Book Philippines, Inc. Prior to finishing his degree in creative writing at the University of the Philippines, Diliman, in 2012 he was a computer engineering student, a ghostwriter, and an iOS Developer working on artificial intelligence for a predictive, pressure-sensing paint app. After the project was frozen, he continued ghostwriting and shifted to creative writing, where he received the Gawad Rogelio Sicat award for the Filipino essay category, and the Gawad Antonio M. Abad award for best undergraduate thesis in the College of Arts and Letters. He is currently exploring the realm of machine learning and poetry to create generative lyrical poetry.